Technische Universität Ilmenau

Fakultät für Elektrotechnik und Informationstechnik

Diplomarbeit

The usage of Ontology Enriched Metadata in a Mobile Context

---

vorgelegt von:	Nikolaus Daniel Sommer
eingereicht am:	29. 12. 2009
geboren am:	14. 09. 1980 in Frankfurt am Main
Studiengang:	Wirtschaftsinformatik
Studienrichtung:	Informations- und Wissensmanagement
Anfertigung im Fachgebiet:	Kommunikationsnetze Fakultät für Elektrotechnik und Informationstechnik
Verantwortlicher Professor:	Prof. Dr. rer. nat. habil. Jochen Seitz
Wissenschaftlicher Betreuer:	Dipl.-Ing. Karsten Renhak (TUI)
Wissenschaftlicher Betreuer:	M.Sc. Timo Itälä (HUT)

Acknowledgement

I want to thank Timo Itälä and Prof. Matti Hämäläinen for giving me a job and, especially Timo, for always giving great support by asking the right questions. Thank you for sharing the spririt that ”it's not about science, it's about trying to change the world”.

I want to thank Karsten Renhak and Prof. Jochen Seitz for supporting me from Ilmenau and being always there when there were questions.

Thank you Ingmar (and IAESTE) for making my whole time in Finland possible.

Thank you Caecilie for spending your time on correcting my thesis while you should be writing yours and your LaTeX support.

I want to thank my parents for supporting me in all these years and also for giving a little pressure at the right times.

Kurzfassung

Diese Diplomarbeit untersucht die Verwendung von Ontologien in einem mobilen Kontext. Herzu werden in einem ersten Teil Ontologien, mobile Endgeräte, Tagging und das Semantic Web betrachtet. Danach wird in zwei Anwendungsfällen die Verwendung von Ontologien in einem mobilen Kontext untersucht:

Anwendungsfall I befasst sich damit, wie es möglich ist unter Verwendung eines mobilen Endgeräts Informationen an einen professionellen Service Provider zu übermitteln. Diese Informationen werden von dem Nutzer mit Metadaten angereichert, die aus einer Ontologie generiert werden. Die Verwendung von Ontologien für das Taggen der Informationen erlaubt eine automatische Weiterverarbeitung der Informationen bei dem Serviceprovider. Hierfür werden die verschiedenen Use Cases und Anforderungen untersucht. Später wird eine Implementierung einer Anwendung vorgeschlagen, wofür ein Service Modell und ein Objekt Modell vorgeschlagen werden. Teile dieses Vorschlags wurden in einem Prototyp implementiert.

Anwendungsfall II befasst sich damit, wie Ontologien dazu verwendet werden können, die Fähigkeiten (z.B. Bildschirmauflösung, unterstützte Dateiformate, etc.) mobiler Endgeräte zu beschreiben. Das Ziel dieser Betrachtungen ist eine Verbesserung der Interoperabilität mobiler Endgeräte, anhand des Austauschs ihrer Fähigkeiten. Zu diesem Zweck untersucht diese Diplomarbeit, wie diese Fähigkeiten unter Zuhilfenahme von Ontologien beschrieben werden können, und in wie fern Reasoning Support implementiert werden kann. Abschließend wird ein Vorschlag für eine Implementierung präsentiert.

Abstract

This thesis is examining the use of ontologies in a mobile context. In the first part it is examining ontologies, mobile devices, tagging and the semantic web. After that it discusses two use cases for ontologies in a mobile context:

Use case I investigates how it is possible to submit information to a professional service provider, using ontologies for tagging. The information gets submitted using a mobile Device. The use of ontologies as metadata allows the receiving service provider an automated pre processing of the submitted information. Therefore this thesis investigates the UML use cases, the requirements and suggests details of an implementation of an application, providing a service model and an object model for the different parts of the application. This recommendation was partly implemented in a prototype, which gets presented as well.

The second use case investigates how ontologies can be used for describing the abilities of small devices (e.g. supported documents, screen size). This research has the aim of providing a wider interoperability of different mobile devices, by allowing a communication based on the abilities of the devices. Therefore the thesis investigates the description of the abilities of mobile devices using Ontologies and reasoning support. Finally it provides thoughts on the implementation of an application for this scenario.

Contents

1 Introduction

1.1 The context of OmaHyvivointi:

One of OmaHyvivoint project (translated to English: MyWellbeing Project, see separate description of Project in Appendix) main goals is empowering the citizen to communicate with professionals as illustrated in figure 1.1.

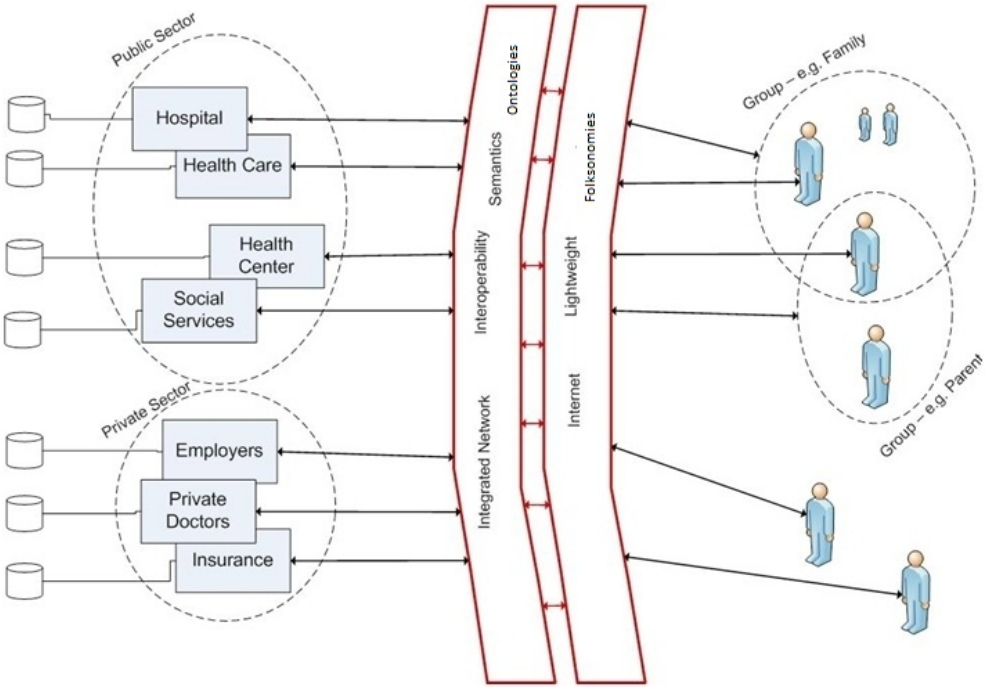

Figure 1.1: Citizens and Professionals.

In an earlier phase of my research for that project I was working on a software for the management of content by assigning metadata to that content. This software allowed a user to store, tag and retrieve information using a PC. In a second version of that software we added multiuser support and allowed basic communications with healthcare professionals.

Considering the omnipresence of mobile devices such as mobile phones, PDAs, etc. an important way of allowing 5 million Finns to communicate with medical profes-

sionals is using those devices as communication platform. Considering that this would create a lot of information sent to professionals, it is important to enrich this information in a machine- understandable way for allowing automated data processing. Using ontologies for tagging is one possible way of doing that. This research can be seen as a logical third step following our prior research.

1.2 Importance

Modern phones offer great possibilities for digital, mobile communication that far advanced simple mobile calls. They include a camera, audio/ video recording features, for tracking positions and Bluetooth for wireless communication with many kinds of other devices. With all these features, these devices are able to create a lot of content, which can be transferred via the internet.

The content produced by phones is mostly unstructured, non machine readable and non search machine indexable. Examples for that content are pictures, videos or audios. These files can easily be published via the Internet using web 2.0 applications like blogs, community platforms etc. For structuring this information it is required to attach metadata to the data. [Mat04] Metadata is defined as ”data about data”, which can have administrative, structural or descriptive functions. The ”collaborative intelligence way” of structuring this information is demonstrated by millions of web 2.0 users. Here users create their own content and describe it using tags. Tags are descriptive metadata, assigned to a resource, and can usually be freely chosen. [Vos07]

For submitting information from a mobile device to a professional service, it is important to be able to add descriptions to the submitted data that can be ”understood” by the receiving system. This allows e.g. doing preprocessing - like submitting the data to the responsible professional, store it or compute some results.

In order to allow the receiving professional’s system to ”understand” the metadata assigned by a user, it is important that both parties agree on a common vocabulary. One possibility for this common vocabulary, typified connections between the used tags, is called ontology [Nag06]. Further explanations on the topic of ontologies will be provided in the next chapter. Different systems might process different content, which may be described by different metadata in different ontologies. Therefore it is necessary to provide an interface for transmitting an ontology, used for tagging to a mobile device.

Another application of ontologies in a mobile surrounding would be that e.g. a service can render a personalized user interface based on the abilities of the device

requesting information. In this scenario a service would request a description of the device, encoded in an ontology. The device would submit that description and receive a personalized view of the service. Other applications are the exchange of data between two small devices, where the devices could negotiate what formats can be processed by the other device.

1.3 Research Question

How is it possible to allow users the use of different ontologies for tagging mobile generated content, under special consideration of the limited features of a mobile device? How is it possible to use similar techniques for submitting the features of a mobile device to a service provider or for exchanging the information about two small devices?

1.4 Explanation of the Research Question

Figure 1.2 shows a possible scenario for the tagging of mobile generated content using ontologies. In this model the user first has to select the content he wishes to upload.

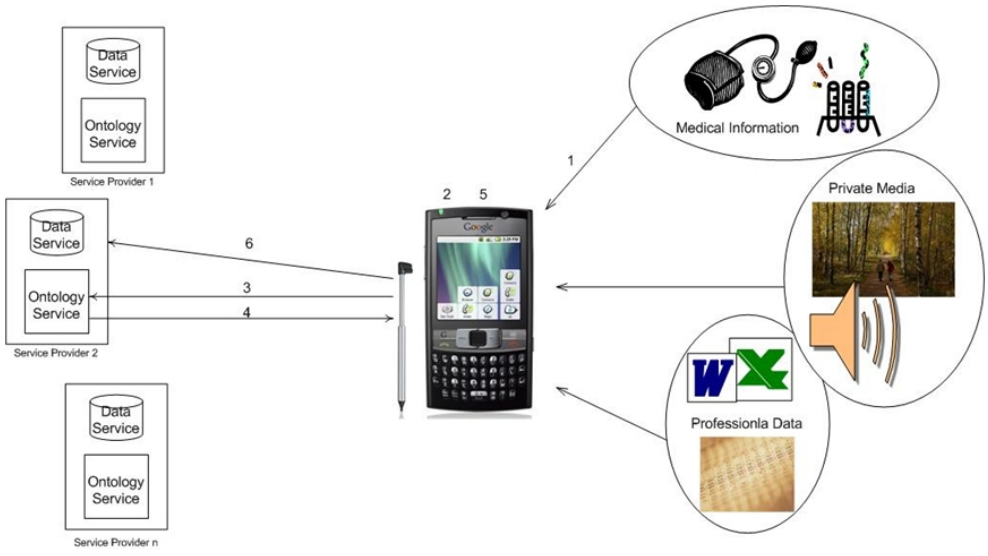

Figure 1.2: Usage of ontology enriched metadata in a mobile context.

Second he has to select the service, he wants to upload the content to. This can be realized using a Service Directory, a Service Bus, etc. [compare [Erl05]]. Third the user has to connect to the selected service and request the services vocabulary, provided

in an ontology. Fourth the service's vocabulary has to be received. This requires a standard for the transmission of ontologies and is one important part of the planned research. The fifth step of that model is to assign metadata to the selected content. This is especially interesting, considering usability issues, e.g. how to browse through huge ontologies using a small screen and how to select the required tags. Last the data has to be submitted to a service, including the metadata.

For the second scenario of submitting information about a mobile device, using an ontology there are a few differences. There has to be a handshake implemented that makes the devices submit the information about their abilities to the other device. There might be less information submitted than in the use case of sending a complex ontology to a device. Suggestions for submitting information about mobile devices' abilities will be discussed later in this thesis.

1.5 Research approach and methology

Most of the research on semantic interoperability will be done by doing literature research. This includes examining companies and software that work in this area as well as standards for ontologies. Afterwards I will point out the special requirements for mobile application development.

Based on that research I will suggest a design for a mobile application, using ontologies for categorizing content.

I will also try to develop (parts of) this application, actually running on a mobile device, that allows the usage of different ontologies.

There are many possibilities considering the development task; it could either be done on one of the large smartphone systems (Android, Symbian, Windows Mobile, etc.) or it would also be possible to create a browser- based web application, designed for mobile devices, that connects to multiple web services.

1.6 Structure

After this introductory part, which gives a basic understanding to the nature of the research problem, I will provide definitions for all relevant subjects in chapter 2. Here I give an introduction to tagging, afterwards I will show how ontologies can be described, especially what formats exist, and how they are used in the context of the semantic web. After that I will present basic knowledge about developing software for mobile

devices. I will discuss the concept of web applications and applications running on a small device.

After describing all required theoretical background, I will perform a requirements analysis for the implementation of an application that allows the mobile upload of data and metadata in chapter 3. Chapter 4 will discuss the possible implementation of that application. Those two chapters will be the main parts of the research performed in this thesis.

In chapter 5 I will describe how I implemented parts of the implementation recommendations. Chapter 6 will discuss the second use case, how to exchange small devices' abilities. In chapter 7 I will provide conclusions and an outlook on further research.

2 Theoretical Background

2.1 Tagging

As the term ”Tagging” is very relevant for understanding the use of metadata, which is a part of this Thesis' title, I am going to give a brief introduction to different ways of tagging. First I will present the process of Tagging itself. Afterwards I will give a definition for metadata and its different uses. Finally I will give a short introduction to social tagging, that is not necessarily connected to the topic of this thesis but as it is a major use of tagging on the internet I decided to describe it briefly.

2.1.1 Tagging definition

Tagging can be defined as ”assigning a Tag to a Resource” [MNbD06]. This process is done by a **Tagger**, who can be a human being, a group of people or a computer program. A Resource can be anything that has the ability of being identified uniquely.

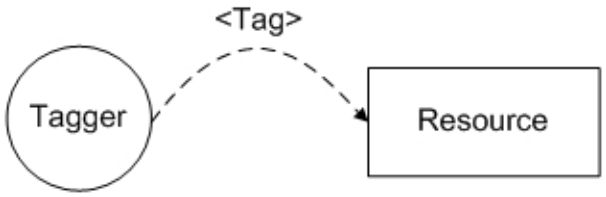

Figure 2.1: Tag, Tagger, Resource. compare [MNbD06].

It can be e.g. a book in a library, having a unique identifier, a website, an image on the internet, a flash film, a set of blood pressure data, etc., all having a for identification. A **Tag** is a word, phrase etc, describing the resource. So a tag can be seen as descriptive **Metadata**. The term metadata will be explained in the next subchapter. [Vos07] Figure 2.1 illustrates the relationship between tags, tagger and resource.

2.1.2 Metadata

Metadata is defined as ”Data about Data” [Mat04]. It is mostly well structured information with a descriptive function. It can describe any kind of data, e.g. books in a library, movies on DVDs at amazon.com, photographs online or in a photo management software at home or medical information in a Hospital Information System. [Mat04]

Metadata can have an administrative, structural and descriptive function [Tay99]:

Administrative metadata can be information about the person being responsible for that data or information about the creator. **Structural metadata** can be information about the quality, amount, and location etc. of data. **Descriptive metadata** is the main use case for Metadata; it is used for describing the content of any kind of resources. [Tay99]

Metadata traditionally has been created by professionals, e.g. catalogers working in libraries, assigning machine readable cataloging records to books or other intellectual goods. This metadata usually is the basis for library catalogs like . This work requires extensive training, as catalogers have to be able to classify also very specific information from all kinds of domains. [Tay99]

For the assignment of metadata in a library context there are plenty of classification schemes: E.g. the ”Dewey Decimal System” or the ”Library of Congress Classification Scheme”, which are guidelines on how to assign metadata. There are as well large predefined vocabularies of terms for describing the content of the materials to classify. [Mat04]

As there is very much content being created in the world wide web every day [LVSC03], it is not possible to categorize all that content by professional catalogers following specified schemes. This makes it necessary to find other ways of assigning metadata to information. Mathes [Mat04] describes two different ways of assigning metadata on the web:

The first one is metadata assigned by the author. For this approach, he sees a problem in motivating the author to describe his own work. There might be also a problem considering the quality of metadata created by the author; the author might have an interest in assigning more popular metadata then appropriate, for making his document more interesting for a wider public [MNbD06]. Even if the author would try to assign appropriate metadata to his document, in most cases he would not be able to do it in the same professional way a librarian might be able to do it.

Mathes [Mat04] describes the second way of assigning metadata by users, which is also known as ”Social Tagging”.

2.1.3 Social Tagging

Social tagging is defined in different ways. One possible definition would be "the manual indexing of different resources that is performed by end-user, not by professionals" [MNbD06]. The assigned tags appear immediately on the internet, without having an editorial staff controlling those tags. As described above, those resources can be anything that can be identified uniquely. Literature mainly sees tags as freely chosen words, some authors [Vos07] also think of a controlled vocabulary that is used for tagging. [MNbD06].

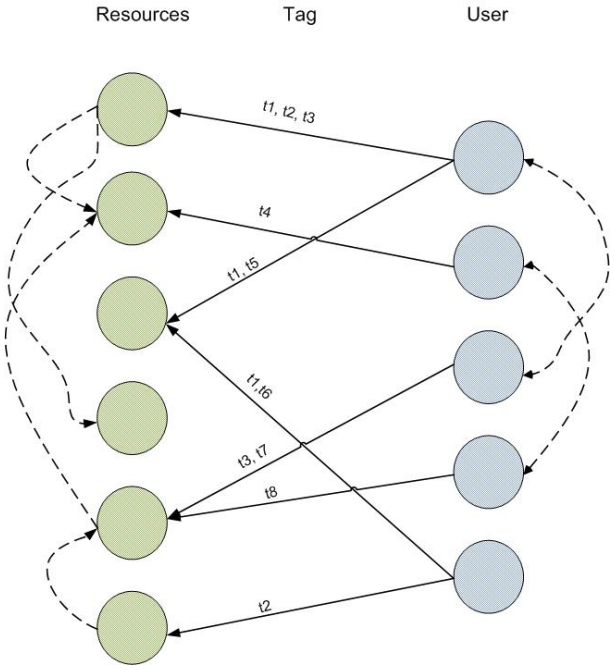

Figure 2.2: Resources, Tags, User. [MNbD06]

Figure 2.2 shows Resources, Tags and User. Here resources can be related to each other (e.g. pictures in the same album, websites that are linked, etc). There can be relationships between users as well (e.g. friends, same interests, common membership of groups, etc.).

For tagging objects on the internet, users need to use a "Social Tagging Platform". This platform is a website, where users are able to assign tags to other websites, pictures or any kind of resource. These platforms allow it as well to search the tags of other users. This offers new possibilities for information retrieval. [GSHB05]

Users might have multiple motivations for tagging. The main reason for the participation of Users is the "future retrieval of resources". "Contribution and sharing" is also an important motivation, as well as "attracting attention" (e.g. for the own website)

or self presentation. [GSHB05]

Yanbe et al. [YJNT07] compare the quality of search in social tagging platforms to the quality of search of classical search engines like google [1]. Their research showed that tagging platforms are much faster in finding new relevant information than classical search engines. This is explained by the operating mode of page rank, the algorithm that is used by google. For having a high rank there are many links to a resource required (link popularity). New information normally does not have many links in the first place. As soon as a few people tag a new resource on a tagging platform it can be found by other users. One example of their findings is that 56 % of the often tagged information on the tagging platform delicious [2] has a page rank of 0, which means that it is not relevant to google. [YJNT07]

2.1.4 Service Oriented Architecture and Web Services

The term "service orientation" has been used for many years. It was used in different contexts and for different approaches. One constant through the different usages has been that is a distinct approach for separating concerns. That means that logic for solving a problem can be better constructed, carried out and managed by decomposing the problem to different pieces. This can be reached by making services encapsulating logic within a distinct context, which is a condition for retaining their independence. This context can be specific to a business task, -entity or any other logical grouping. [Erl05]

One main goal of SOA is that it is a **business and technological** approach and methodology. It enables business to make business decisions supported by technology, not making them determined by technological constraints. [HBB06] SOA also allows the integration of legacy applications (e.g. applications running on old systems) by implementing an interface.

A service is offered by a provider to a consumer through its interface. An interface describes the contract between the provider and the consumer. It defines what the provider is obliged to do on behalf of the service consumer. It can be seen as functionality that must be specified in the business context and in terms of the contracts. Implementation details should be not revealed and the implementation does not have to be automated. [AHMS06] In other words: the consumer does not need to know anything about how the service (software, human activity, etc.) solves the problem.

[1] http://google.com
[2] http://del.icio.us

A service normally implements support for a business process. A business process is a set of activities that is initiated by an event, transforms information or materials, and produces an output. [AHMS06]

For understanding the whole picture, Allen [AHMS06] defines the different parties in the process of implementation and usage of SOA as follows:

A **supplier** implements a service. This may involve the implementation design and/or the actual execution of services. A **provider** agrees with a customer on the kinds of services provided. This agreement is called Service Level Agreement and defines measurable levels of the services a provider must achieve as well as the terms of service the customer must comply with. The provider can be the service supplier as well. A **customer** agrees on a with the provider and makes sure the required business tasks are implemented. The customer also provides funds for using the services. A user (**service consumer**) actually uses the service, according to the terms negotiated by service provider and customer. [AHMS06]

Figure 2.3 illustrates the relationship between service provider and service consumer. In this case multiple service consumers request the service of multiple service providers. The providers are addressed through a common Service interface, so it is not visible for the consumers which provider actually performs the required operations. [AHMS06] A

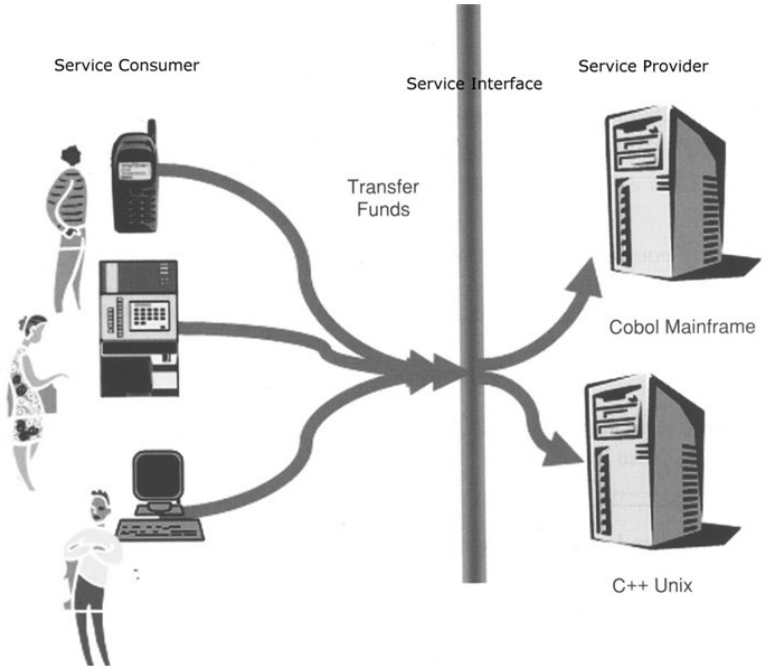

Figure 2.3: Service Provider - Service Consumer. [AHMS06]

service can also request the assistance of other services. So it can be service provider

and service consumer at once. Figure 2.4 illustrates how a service can call another service for assistance. Service B could be an intermediary service that acts as a routing service. It would also be possible, that service C is an assistance service doing mathematical operations, required for solving service B's task (e.g. conversion of different currencies). [Erl05] The Messages sent from one service to the other have to be in

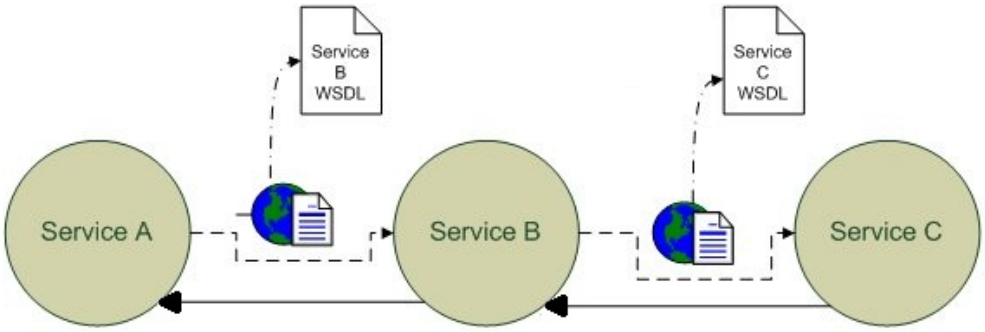

Figure 2.4: Services connected. Compare [Erl05]

a standardized format. Some different formats will be discussed in later chapters.In order to realize an automated, self managed SOA, it is necessary to make the services work together. SOA offers the concept of the Enterprise Service Bus, as an essential architectural element. The ESB is a core intermediary, which ties services together into componentized, logical sets. The main infrastructural services provided by an ESB are transport, routing and security. It acts as the intelligent, distributed, transactional and messaging layer for connecting applications, diverse data, and other services. The actual implementation of an ESB can be realized using existing EAI Software (e.g. SAP R3), classical messaging, or platform specific components. [BMF06]

There are several design principles when designing a SOA. They are: Loose coupling, abstraction, reusability, autonomy, statelessness, discoverability, composeability and formal contracts. For more details review [Erl05].

Web service principals will be discussed in detail later in this thesis. At this place the author just wants to point out the relationship of SOA and web services. SOA is reliant on web service technology. Web services also use the concepts of a service description and messaging, introduced above. One main difference between SOA and web services is that designing SOA includes identifying the key business processes and implementing those in services. Web services on the other hand are just simple applications that are used by other applications and perform different tasks. There is no universal integration or orchestration involved as it is done in SOA using the . Web services are discovered using a service registry, where services provide all necessary

information for their implementation. [HBB06]

Details about the communication of web services will be provided in later chapters.

2.2 Ontologies

As the term "Ontology" is part of the thesis' title, it is necessary to define different aspects of ontologies. First I will give a general definition of the term; afterwards I will differentiate ontologies to other terms that refer to similar concepts. In the next two subchapters I will give examples for the use of ontologies by presenting the idea of the semantic web that is based on the use of ontologies, followed by a subchapter about other uses of ontologies.

Afterwards I will present languages for the representation of ontologies like and that can be used for submitting ontologies from one device to another. Finally I will discuss ways of sharing ontologies. Those two last subchapters are especially relevant for the implementation of ontologies into a software application, discussed in the following chapters.

2.2.1 Definition

In philosophy, ontology is defined as "the theory of the nature of being or the kinds of existents" [Ehr07], first discussed by Socrates and Aristotle. Socrates introduced the notion of abstract ideas, (e.g. hierarchy, class-instance relationship) and Aristotle added logical associations. [Ehr07] More than 2300 years later, Taylor [Tay99] defines Ontology as "in the field of artificial intelligence, a formal representation of what, to a human, is common sense". She goes on defining ontology for the field of natural language processing, which is a Computer analysis of written or spoken word, in order to interpret meaning, allowing the computer to "understand" and "reply". Her definition for ontologies is a "formal representation of language, including realities of such things as grammar, semantics and syntax". [Tay99] These definitions can be seen as problematic, as they use terms as "common sense for a human" and "understand". Computer systems are able to process information, but not to understand it.

In literature the definition of Gruber [cited after [AH04]] is widely accepted [e.g. [Ehr07], [AH04]]:

"An Ontology is an explicit and formal specification of a conceptualization." The term conceptualization can be understood as an abstract model of some real world phenomenon. Explicit means, that the types of concepts and the constraints on their

use are explicitly defined. [Studer 1998 after [Ehr07]]

This definition is sometimes extended:

"An Ontology is an explicit, formal specification of a shared conceptualization of a domain of interest" [Ehr07].

The additional terms here are "formal", which refers to machine-readability, "shared", which refers to a common agreement (at least by a group of people) on an ontology and "domain of interest", which means that ontologies always have a limited scope and that it is not useful to model the whole world. [Ehr07]

2.2.2 Differentiation to other Terms

There are a few terms that need to be differentiated from ontologies as they describe some different concepts for connecting representations of real world objects. These are associative relations (using an authority file), monohierarchical relations (using classification or taxonomy), multihierarchical relations (using a thesaurus) and finally typed relations (using an ontology). [Vos07] **Associative relations** are used, where concepts or terms are related in any other way than hierarchical or equivalent. They are also referred to as "see also relationship". Any types of relations are possible here. [Vos03] The term "swimmies" could be e.g. related to the term "children" as well as the term "diaper". "Diaper" on the other hand could be related to "old people" as well. This would not say anything about a relationship between all of those terms. So "diaper" is not necessarily related to "swimmies" or "old people" to "children". "Related terms" are a definition for associative relations. [Vos03] An authority file is mainly used for cataloguing in libraries. Here all relevant information on an author is stored, e.g. his books, but also books about him. Those files contain associative relations e.g. between an author and books, subjects, etc. [Tay99] A **monohierarchical relation** is a relation where every class has one upper class (except the top class, which is the root without any upper class). In a **multihierarchical relation** every class can have any number of upper classes [Voss 2003]. Figure 2.5 illustrates this differentiation. Taxonomy is a way of using a monohierarchical relation, where all objects are represented in a tree structure. This is typically used for classifications. A thesaurus on the other hand allows one class to have more than one upper class. [Vos03] Therefore a thesaurus is not usable for a hierarchical or tree classification. [Kwa99] A **typed relationship** can be mono- or multihierarchical. The difference is that relations have a type like "is a" "lives with" "likes to eat" or any other type. This is realized in using ontologies.

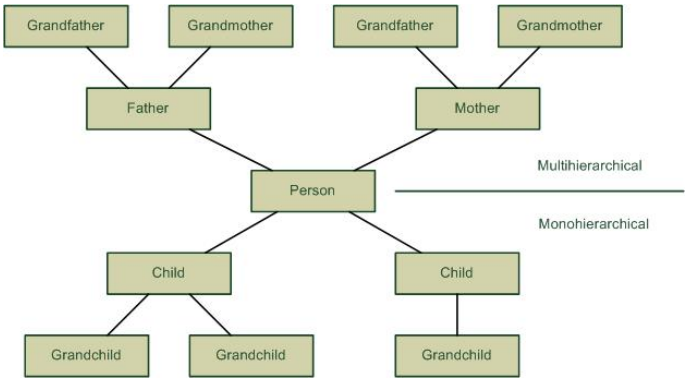

Figure 2.5: Multihierarchical vs. monohierarchical relations. [Vos03]

2.2.3 The Semantic Web

2.2.3.1 The World Wide Web

Originally computers were used for calculating equations in a scientific or business field. Later they were used for different office applications. The World Wide Web has extended the use of computers to being entry points to the global information highways. They can be used for viewing any kind of content, even if it is stored in databases and generated on the fly. [AH04]

The Internet was born in 1978 with the creation of TCP/IP, which have been the globally used protocols of all computers using the internet. is the "Transmission Control Protocol", which is responsible for verifying the information transmitted from Server to Client. is the "Internet Protocol", which is responsible for forwarding information from node to node, based on a four byte destination address and allows the process of routing data through different networks and nodes to the final destination. With these protocols it was possible to connect separated networks into a network of networks, which is now called the Internet. [AS04] In 1991 Tim Berners-Lee introduced the "Hypertext Transfer Protocol" for linking web documents, the "Hypertext Markup Language" for formatting web documents and the "Universal Resource Locator system" for addressing und identifying web documents. Together with the availability of web browsers this was the start of the World Wide Web. [AS04]

2.2.3.2 Limitations of the World Wide Web

Tim Berners-Lee describes the limitations of the World Wide Web in his paper "The Semantic Web" as follows: "The essential property of the World Wide Web is its universality. The power of a hypertext link is that "anything can link to anything."

Web technology, therefore, must not discriminate between the scribbled draft and the polished performance, between commercial and academic information, or among cultures, languages, media and so on. Information varies along many axes. One of these is the difference between information produced primarily for human consumption and that produced mainly for machines. At one end of the scale we have everything from the five-second TV commercial to poetry. At the other end we have databases, programs and sensor output. To date, the Web has developed most rapidly as a medium of documents for people rather than for data and information that can be processed automatically." [BLHL01]

The hyperlinks of the web represent structures of meaning, but those structures cannot be navigated properly as they can change every second. The only way of using those structures is the use of a search engine that analyzes the links and sorts documents by relevance. Under normal circumstances content providers don't have any influence on their position in those search rankings. [AS04]

HTLM also does not provide any proper interface for complex machine to machine communication. In many cases structured data gets pulled out of databases and automatically rendered in a human understandable, graphical format. It is a very complex task for a machine to parse those HTML files and transform them back to a machine computable format. [AH04]

2.2.3.3 The Conclusion: The Semantic Web

As a result of the limitations of the World Wide Web, described by Berners-Lee in "The Semantic Web" he develops the following scenario as a vision that gives a good understanding about the basics features of the semantic web "The entertainment system was belting out the Beatles' "We Can Work It Out" when the phone rang. When Pete answered, his phone turned the sound down by sending a message to all the other local devices that had a volume control. His sister, Lucy, was on the line from the doctor's office: "Mom needs to see a specialist and then has to have a series of physical therapy sessions. Biweekly or something. I'm going to have my agent set up the appointments." Pete immediately agreed to share the chauffeuring. At the doctor's office, Lucy instructed her Semantic Web agent through her handheld Web browser. The agent promptly retrieved information about Mom's prescribed treatment from the doctor's agent, looked up several lists of providers, and checked for the ones in-plan for Mom's insurance within a 20-mile radius of her home and with a rating of excellent or very good on trusted rating services. It then began trying to find a match between available appointment times (supplied by the agents of individual providers through

their Web sites) and Pete's and Lucy's busy schedules. (The emphasized keywords indicate terms whose semantics, or meaning, were defined for the agent through the Semantic Web.) In a few minutes the agent presented them with a plan. " ... "and it was all set" [BLHL01]

This Scenario involves the following devices: An entertainment system, a phone, a handheld and computer. It also involves a few software applications: A service that mutes all audio devices in the surrounding of the phone, Lucy's semantic web agent, the doctor's agent, an agent that supplies a provider list for the treatment, the insurances agent, a geographical agent, the providers' agents and Lucy's and Pete's schedule.

This procedure is not possible in the World Wide Web. It requires the collaboration of many agents and services that are able to perform negotiations without human interaction. Agents are software programs that work autonomously and proactively. They evolved out of the concepts of object oriented, concept-based Software development. [AH04]

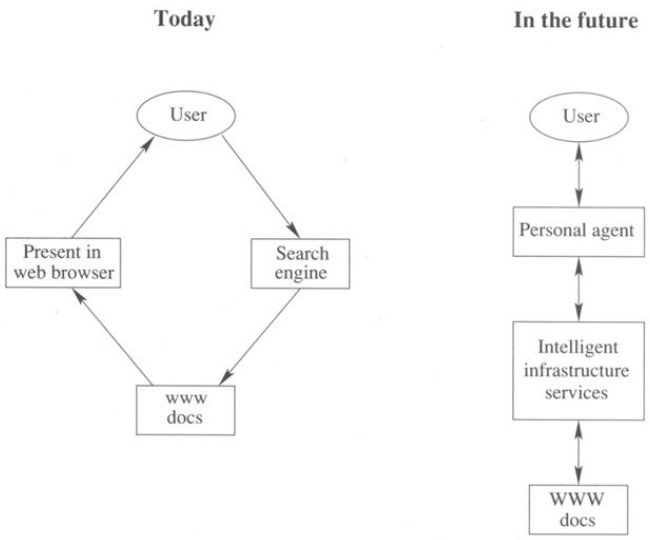

Figure 2.6: Intelligent personal agents. [AH04]

Figure 2.6 presents the process if search, using an intelligent personal agent, compared to a standard web search. Using a web search, a user has to search for information using a search engine. The located information gets presented in a web browser and the user has to read and evaluate the information by himself. He also has to perform the required bookings, tasks, etc. The intelligent personal agent on the other hand will receive a task, preferences and information from the user, and will automatically

search information, communicate with other agents and evaluate information, depending on the user's preferences. Finally it will present the best answers to the user for allowing him a decision, based on the main facts. For enabling agents to perform the tasks, described by Berners-Lee above, it is first necessary that they are able acquire information considering the related subject. Second they need to be able to communicate in a machine understandable language and are able to evaluate the acquired information. Third they have to be able to select the best solution to the presented question. [AH04]

Those requirements are implemented to the semantic web: Metadata, which was already discussed in this chapter, provides structured data that can be easily parsed by the agents. For giving a meaning to that metadata and for communicating to other agents, agents can use common ontologies. For drawing conclusions agents are equipped with a set of logic, which allows them to process the retrieved information. For finding the appropriate agents there is a need for a special service that describes the abilities of existing services as well as their URI. This process is called service discovery and it responsible for listing Agents/ Services and describing them in a way that other agents can "understand" what functions are offered and how to use them. The service discovery can be seen as "Yellow Pages" of the semantic web. It is also ontology based for universal interoperability. [AH04]

2.2.4 Ontologies in other fields

Ontologies can be used in any field where it is necessary to organize information and model it using relationships. For giving a quick overview, I will briefly describe two examples for other scenarios.

Goncalves et al. [GFF01] uses Ontologies for modeling the differences between incompatible library information systems. This allows providing an ontology based approach for mapping those systems for performing search operations on multiple, otherwise incompatible, libraries using the ontology based middleware [GFF01].

There is also research related to ontologies for supporting the business. Osterwalder and Pigneur [OP02] introduce an approach where a business framework is based on the modeling of an ontology. This ontology has four main pillars: The products and services a company offers, the infrastructure and the network of partners, relationship capital the company creates (based on customers, revenues, etc.) and the financial aspects. This ontology allows modeling the whole company and its surroundings and is the base of a three layer model for simulation. The second level of this framework

is the ”Measures Level”, which helps identifying relevant indicators. The 3rd and highest level is the ”Dynamic Equation Level”. This level provides an interface for running simulations based on the ontology and previously defined indicator values. The authors describe the main goal as ”a sort of e-business model flight simulator”, that allows managers ”to play around in a risk free environment”. This modeling-simulation approach can also be used in other domains. [OP02]

2.2.5 Laguages for the representation of Ontologies

2.2.5.1 XML

The main languages for representing ontologies are XML based. As XML is a well known topic, I decided to describe it very shortly for giving a better understanding of the actual languages used for representing ontologies.

XML stands for Extensible Markup Language. It is a set of rules dividing documents in parts by using semantic tags. Those tags allow it to identify the different parts of the document. is not just a markup language it is a Meta Markup Language. That means it is possible to design any kind of markup language using the XML syntax, which allows it to introduce all tags required for the current situation. for example is a markup language with a fixed set of tags. It is not possible to state anything not supported by those tags. XML on the other hand provides a framework for designing new Markup Languages that allow the exchange of structured data with any kind of vocabulary required. HTML is a Markup Language for describing formatting. XML on the other hand describes structure and semantics. [Har04]

```
<?XML VERSION ="1.0">
<PERSON>
    <NAME>Niko</NAME>
    <AGE>27</AGE>
</PERSON>
<PERSON>
    <NAME>Oma Lore</NAME>
    <AGE>86</AGE>
</PERSON>
</XML>
```

Figure 2.7: XML Document.

Figure 2.7 shows an example of an XML document. This document contains in-

formation about 2 persons and their age. A well formatted XML document has no overlapping tags. If we would send this XML Document from one Service to another, both services need to know the tags <person>, <name> and <age>. If the receiving service had no implementation of the tag <age>, it would just ignore it. If the sender had not an implementation for this tag, there might occur an error at the receiver. For ensuring that XML Documents contain all required information, it is possible to define this information using Document Type Definitions. DTD allows to define the structure of an XML Document with mandatory and optional elements, cardinalities and formats of elements. So DTDs provide the framework for defining a new markup language, using the meta markup language XML. By knowing what elements are required, it is possible to guarantee an exchange of data between different services, as the designers know what elements have to be implemented in their services. [Har04]

2.2.5.2 RDF / RDFS

The acronym RDF stands for Resource Description Framework. RDF is the first language especially developed for the semantic web. The first recommendation for RDF was released in 1999, for the RDF Schema in 2000. RDF is built on XML terminology. It uses his own grammatical representation which consists of the triple Subject, Predicate and Object. This relationship is viualized in figure 2.8. [AH04]

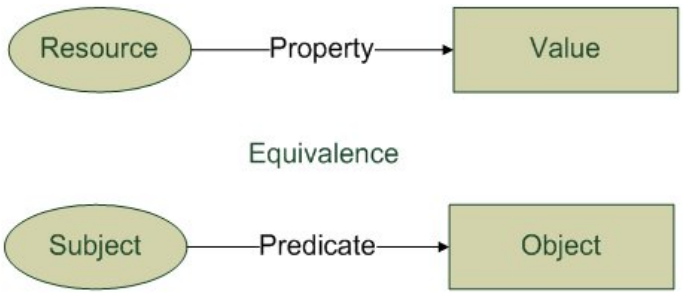

Figure 2.8: Graphical Statement for RDF. [AS04]

If we take an example statement where a ”car” [subject] has the ”brand” [predicate] ”Skoda” [object], it would be possible to note it with the triple (Subject, Predicate, Object) (car, brand, Skoda). Another statement would be (Car, owned by, Hartmut) and (Hartmut, father of, Niko). If we would draw these three example statements as a graph, there would be two subjects: ”Car” and ”Hartmut”. ”Skoda” and ”Niko” would be objects. In the ”owned by” relationship, Hartmut would be an object as well. If we would add (Niko, has birthday, 14.09.1980) Niko would also be used as a subject.

Each RDF Document should include a namespace. Those namespaces are usually RDF documents where resources are defined that can be imported by the current RDF document. The rdf:about statement is equivalent to the meaning of an ID argument. It implies that this subject has already been defined at another place, like the imported namespace. A set of RDF statements just forms a large graph, where things are related to other things through properties. [AH04]

```
...
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:carowner=http://ownership.com/carowner-ns#>
<rdf:Description rdf:about=http://ownership.com/carowner-ns/#car1234>
    <carowner:carbrand>Skoda</carowner:carbrand>
    <carowner:owner> http://ownership.com/carowner-ns/#hartmut </carowner:owner>
</rdf:Description>
</rdf>
```

Figure 2.9: RDF modeled relationship.

Figure 2.9 illustrates a small RDF file, where a car gets an owner and a brand assigned as properties. The namespaces xmlns:rdf and xmlns:xsd give standardized links to the RDF model, syntax and schema specifications. Xmlns:carowner gives a namespace where car1234 and hartmut are already defined. That allows us to reference them in our RDF file. The brand Skoda is not defined so it given as a string value. The properties carbrand and owner are defined as well.

RDF provides a simple way of describing information. The meaning of that information is described using RDF Schemas. An RDF Schema allows expressing classes. Those classes can be related to each other using classes, subclasses, properties and the concept of inheritance. By designing RDF Schemes, it is possible to model hierarchies and relationships. Those are used later for computing the information collected in RDF Files, that link to that RDF Scheme, by following rules for computing answers to questions. Unlike XML Scheme it does not give any information about the syntactical appearance of the RDF description [FLB+07]

There are certain limitations considering RDF (Scheme): e.g. it is not possible to define properties of properties, necessary and sufficient conditions for a class membership or equivalence and disjoint relationships of classes [AS04]. Those limitations made it necessary to design a new language with a greater expressive power for the representation of ontologies.

2.2.5.3 OWL

As mentioned above, RDF is used for describing relationships (Predicates) between Subjects and Objects. RDFS is used for defining relationships between classes. OWL also describes classes and properties as well as their relationships. So it is very similar to RDFS and RDF. OWL offers much richer statements for expressing those relationships. As RDF Schema is compatible to OWL and there are RDFS elements within the OWL element set. [AS04]

There are three different versions of OWL, with different sets of functions, which I will briefly mention. The most limited is OWL Lite, which does not offer disjointnes, enumerated classes or arbitrary cardinality. It is limited but therefore easier to use. OWL DL has more features but does not allow that a class can be used simultaneously as a collection of individuals and as an individual. This shall ensure a good description logic, but on the cost of not full compatibility to RDF. OWL Full offers the whole spectrum. Further considerations will reference OWL Full. [AH04]

The header of an OWL document is similar to an RDF document. It adds a xmlns:owl for referencing to the OWL namespace. It is also possible to use owl:import for importing different ontologies. The class elements on the other hand are far more sophisticated than RDFS elements. Classes in OWL are very adaptable. Every class is a subclass of the predefined class owl:Thing and the class owl:Nothing is empty. Classes can be defined as equivalent or disjointWith. It is also possible to unify classes for building a new class. [AS04]

There are some properties implemented that can be used defining Classes [AS04]:

Transitive: `P(x,y) ^P(y,z) implies P(x,z)`

Symmetric: `P(x,y) if P(y,x)`

Functional: `P(x,y) ^ P(x,z) implies y =z`

Inverse: `P1 (x,y) if P2(y,x)` e.g. Parent is inverse to child.

[AS04]

These properties can be combined. It is also possible to define cardinalities or unique properties. Instances of OWL classes are declared in RDF [AS04]. This expressive power allows it to create more sophisticated class structures than in RDFS. This can be used for a larger support for reasoning and artificial intelligence, using rule based predicate and description logic through the mapping of OWL to those logics.[AH04].

2.2.6 Sharing Ontologies

As mentioned before, ontologies are mostly stored in XML documents using RDF or OWL. Using OWL, those ontologies are extendable by referencing other OWL resources in a different OWL file. Those referenced ontologies need to be accessible through the internet, which requires an URI. If two different parties use two different ontologies for modeling similar circumstances, they might not necessarily use the same terms for the representation of those circumstances. It is also possible, that the same terms are used for different circumstances. Both scenarios result in non compatibility. This problem is not only relevant for ontologies it is a general problem of outspread knowledge representation designed by different entities or users. If Bobs personal Agent tries to communicate with another Agent e.g. for buying a book about "Java" - because Bob wants to go on vacations to this island - the other agent might know the term "Java" as a programming language and deliver wrong results. There are approaches for aligning different ontologies. Figure 2.10 illustrates the process of ontology alignment.

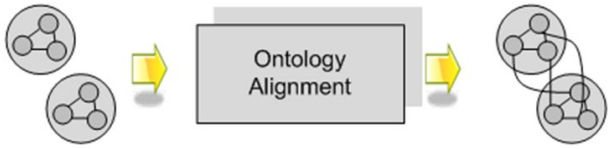

Figure 2.10: Ontology Alignment [Ehr07].

Here two or more ontologies are analyzed for elements with the same intended meaning and finally those elements are recognized and connected. This procedure is based on exploring the semantics of the ontologies which improves the results of the combined ontology compared to simple label-based approaches. Ehrig describes different scenarios, where ontologies are combined automatically and others where the user is involved in the process of alignment, which improves the results. [Ehr07]

Another approach for assuring semantic interoperability is the concept of an "Upper Ontology", developed by the Standard Upper Ontology Working Group of IEEE. The goal of this Upper Ontology is to "support computer applications such as data interoperability, information search and retrieval, automated inferencing, and natural language processing." [Sch03]. The purpose of an Upper Ontology is to give a scheme in which very general terms are modeled. It is not used for modeling a domain specific ontology, but being abstract enough for allowing different domains to use the Upper Ontology as a guideline and master. This scheme allows application developers to extend the ontology. It plays the role of a common vocabulary, which enables applications to at least communicate on the implemented minimal vocabulary of the upper Ontol-

ogy. It finally enables the owners of existing applications, using an ontology based on a specific domain, to map concepts to the upper ontology. This provides support for the above described process of Ontology Alignment [Sch03].

2.3 Implementation of Applications on Mobile Devices and mobile Web Services

As this thesis is about using ontologies for tagging on mobile devices, this chapter will give an introduction to mobile devices and the implementation of software on them. First I will introduce the term "mobile devices" as well as their pervasiveness. Here I will point out the special requirements for developing software in a mobile context. Afterwards I will present the main differences to other platforms, like PCs. Finally I will describe special requirements for developing mobile web services.

2.3.1 Moblie Devices

Currently there are 100 Million active cell phone contracts in Germany. That means a market penetration of 105%, which means more than one contract per citizen. The market penetration of landlines on the other hand is just 83% [bmw08]. This numbers show that mobile communication is ubiquitous.

Modern smart phones offer far more functionality than just receiving calls or sending text messages. They provide fast internet connections, mobile browsers, offer development platforms that allow the creation of software for those phones and also allow the connection to other devices using Bluetooth or IRDA. There are also other mobile devices offering internet connectivity. Those devices realize that connection using WIFI, Bluetooth, etc. [Pas05]

Relevant devices for this thesis need to be able to run software that was specially developed for those devices, or at least need to run a mobile browser. It is also required to have the possibility to establish an internet connection. Mobile devices (small devices, simultaneously used), relevant for this thesis are: Portable devices (like smart phones or PDAs) that are able to install new Software and allow the connection to the internet. A Smartphone can be defined as "A large-screen, data-centric, handheld device designed to offer complete phone functions whilst simultaneously functioning as a personal digital assistant." [Analyst House Gartner after [Bes06]] Pashtan [Pas05] examined the market situation of mobile Devices in 2005. He contested that modern phones mostly have larger color screens, compared to phones a few years ago, that just

provided small black and white displays. Nevertheless the screen is still small compared to a personal computer. They provide functionality for watching videos, offer fast internet connections through 3G and offer multimedia functionality for playing and recording music, videos and pictures . [Pas05]

Due to the rapid development in computer systems and mobile devices the Author will present the capabilities of two state-of-the-art smart phones.

The Apple "iPhone 3G" has a 3.5" Multi Touch Display, a camera for recording videos and pictures, a speaker for listening to sounds, UMTS, HSDPA, GSM, Wi-Fi, and Bluetooth for connecting to the internet or other devices, assisted GPS, that can be used for localizes services and up to 32 GB of memory for Applications, Music, Videos or any other Data. It gets shipped with a mobile browser, that allows to see most of the standard websites, except those using AJAX or Flash." [app09c]

Developing software for the iPhone is adapted to the needs of a user in the mobile environment. It is realized using the iPhone SDK. The lower levels of the system architecture are similar to those in Mac OS X. The higher levels provide media support (2D and 3D drawing, audio and video), object oriented support (for collections, file management, network operation) and visual support (for the creation of applications like windows, views, controls etc.). While developing applications it is recommended to use a top down approach, realizing as many functions as possible in the higher layers. iPhone OS does not support multitasking, so the running application is the only one using system resources except some low level daemons, like System Services, the phone application, or audio playback running in the background. The advantage is to give the main application as many resources as possible. The disadvantage is that it is not possible to design software running in the background. iPhone SDK offers new events, specially designed for that device like multi touch events for zooming or double touch for other features. [app09b]

Another phone using a different concept is the "HTC Touch Pro 2" that provides a full QWERTY- Keyboard, which is located under the screen and can be pulled out. This makes entering text a lot easier. The connection and multimedia features are similar to those, described for the iPhone. The main difference is the operation system used, Windows Mobile 6.1. [HTC09] It allows multitasking and gives developers more possibilities for developing their applications, also running in the background. [Mic09] It supports the installation of Opera Mobile, a browser that allows the support for JavaScript, AJAX and Flash light, for using some lightweight features of Flash technology [int08]. This is a big advantage for creating mobile websites and mobile web service.

Developing software for Windows Mobile is supported in three ways. It is possible to use native code for creating high performance applications with direct hardware access, programming in Visual C++. The second way is usage of managed Code (C# or Visual Basic) for writing software on a higher abstract level. This gives the developer help for connections to web services and provides ready to use graphical interfaces. The third way is developing server side code, which is located on a server and allows guaranteeing connections to other services [msd09]. Sun introduced the Java Wireless Client for Windows Mobile that allows running Java applications on Windows Mobile Devices [SUN08]. J2ME is the Java Language for mobile devices. The advantage of J2ME applications is that they can be run on various mobile devices as they use a virtual machine that allows creating software not depending on the actual operating system but on the existence of a virtual machine for that system. This enables developers to design software for many different devices programing just one application in Java. [Pir02]

Comparing the development abilities it is to say that the Windows platform offers a wider range of development tools, as there are possibilities to develop native code, that can also be run in the background as well, and Java code. The distribution of software is also much easier as it is possible to provide installer programs to any user, on the iPhone the software gets controlled by Apple before it is possible to provide it to users using Apple's proprietary store.

2.3.2 Differences to other platforms

The following section will discuss the development of standard software. For special software, like computer games or other software with high requirements -considering for example the graphics adapter - there are more differentiations to make. As this kind of software is not part of this thesis, I will not get into that topic.

Developing standard software for standard PCs gives the developer some expectations of the PC the Software is supposed to run on. He can normally expect to have a Screen with a resolution of at least 1024 x 768 pixels, an internet connection, a keyboard and a mouse for the input of data. If he develops MS Windows compatible software it can be run on almost all computers, a few of them requiring emulation software, e.g. for MAC OSX or Linux [Kei08]. All modern computers have enough computational power for running standard software.

While developing software for smart phones, the expectations of the developer are different and not clear at all. There are phones with large touch screens with high

resolutions like described above, but also phones using small screens and a standard phone Keypad for the input of data. The internet connection is not always available as plenty of users are not willing to pay for expensive data plans. Some phones support multi touch gestures for input, some do not. It is also not clear if the phone, the software is supposed to run on, will have a camera, Bluetooth for connections to other devices, WiFi for fast internet connections, or a proper speaker for playing sounds.

There are also a lot of different operating systems, requiring different source code for the applications. The above described use of J2ME allows targeting a broader range of devices but there are functional limitations compared to native software. [Pir02] All these limitations require that the developer specifies the exact requirements for the phone the software is supposed to run on.

One other difference to developing Software for standard PCs is in the way users actually use the application. They are on the road, want to get information quickly and they do not have time for browsing through complex navigation structures. Mobile applications should be designed clearly and easy to use. Providing a simple layout that enables the user to find the required information is one of the main design principles. [App09a]

2.3.3 Web Services on mobile devices

There are mainly two different options of allowing a user to run software on a mobile device. The first option is to develop an application that actually runs on the device, the second way is develop a web application that is running on a server and displayed in the devices browser. XXX The problems of developing applications for mobile devices, like different operations systems, different screen sizes and different ways of input of data are described above.

A web application is software, which is running on a server and displayed in a browser that renders the user interface. Encoding websites on a standard PC is normally done using HTML There are several Markup languages for encoding websites for display on a mobile device with different abilities: [Pas08]

Compact HTML was created in 1998. It is a standard close to HTML. It uses all features standard HTML uses but JPEG image, tables, image maps, multiple character fonts and styles, background color and image Frame Style sheets. This allows the display of limited mobile websites even on small screens. [Kam98] The possibilities of developing sophisticated software based on compact HTML are limited.

Wireless Markup Language 1.3 was defined by the WAP forum and released in

February 2000. Content in WML is organized in cards that can contain formatted text, input elements, select elements and fieldset elements that act as organizational containers. Cards are organized in so called decks, that contain cards that a user might use in a coherent context. It is possible to navigate between cards using <do> and <go> statement. In general it is to state, that the possibilities of building complex applications are very limited by the design functionalities of WML 1.3. [Ope01]

Extensible Hyper Text Markup Language Mobile Profile is built on top of XHTML Basic. is an XML compatible version of HTML with the same expressive power as HTML but also the need for being well formatted as XML. XHTML adds some mobile specific modules that allow e.g. the use of forms, style sheets and different input modes. [Pas08]

A mobile browser is responsible for the interpretation and the display of those markup languages. There are a lot of different browsers available that support different standards. 1.x is supported by the majority of browsers. Other languages are fully supported by some browsers, partly by others. Modern browsers even support the display or HTML pages. The quality of the user experience here is depending on the devices' screen and the implementation of navigating through pages that are mostly designed for larger screens like on a standard computer. [Pas08]

The latest version of Opera Mobile even supports parts of Flash technology and AJAX for interaction with elements that are generated on the client side. [Ope09]

One other problem is illustrated by Passani in Figure 2.11.

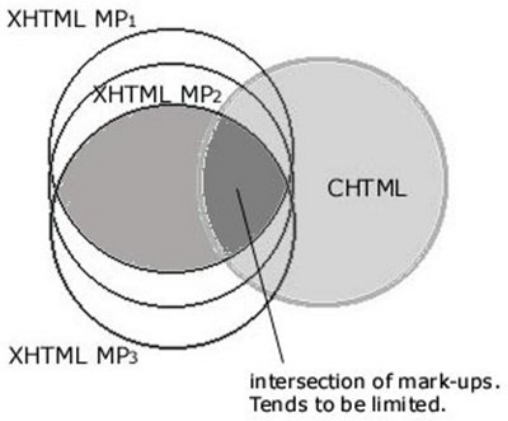

Figure 2.11: Intersection of different Mark up Languages. [Pas08]

In theory applications should be interpreted by all devices in the same way. Due to different browsers and different XHTML versions or (another Mobile Markup Lan-

guage) full support is not always provided. So there are different interpretations of the same code, and there is just a small intersection of mark-ups that are interpreted equally by all devices. New browsers might limit the intersection even more. [Pas08]

Passani's approach for solving that problem is introducing WALL, a Library to Multiserve Applications on the Wireless Web. WALL is a Software that generates Code on the Fly, depending on the requesting device. It uses the database of that contains the capabilities of most devices. WALL detects what markup language is preferred by the requesting device and generates code in that language. The advantage is that all devices receive a code they can display in the correct way. This is realized by using a Java application that checks the used user agent, queries WURFL for the agents preferred language and generates a document, understandable for the client. [Pas08]

Even though the ability of translating sites from WAP 1.0 to modern HTML in high resolutions, using and Flash for Opera's mobile browser, is not implemented, this procedure allows generating a web site using the optimal Markup Language for a lot of mobile devices. Use Case 2, on the interoperability of mobile devices, will give a deeper insight to this approach.

3 Requirements Engineering

This chapter tries to solve the question what requirements need to be fulfilled for developing an application that allows the transfer of data and ontology enriched metadata from a mobile device to a professional service provider. The requirements engineering is an important part of any software development process. This is the reason, why it is also necessary for this theoretical development of an application, allowing the use of ontologies as metadata for the tagging purpose. This is an important part of trying to solve the research question. The use of ontologies on a mobile device is the main part of this section. [Som04]

The Requirements Engineering in this chapter is done following the methods of the Package, which is presented in the first section of this chapter. In the following subchapters I will follow the suggested steps of the SRS Package for allowing a structured requirements engineering.

3.1 Method: The SRS Package

SRS stands for Software Requirements Specification and can be seen as a logical package that helps creating a logical structure for analyzing the requirements for a software application. It serves as a basis of communication among all parties, such as developers, external groups, users or other stakeholders. It also represents an agreement among those parties. Furthermore it serves as a reference for the project manager who can compare the software to the SRS documents. Lastly it is used as an input for design, implementation and testing groups. [LW00]

The relationship between the different parts of requirements engineering is presented in Figure 3.1:

The Requirements Pyramid demonstrates the relationship between the different parts in the software development cycle: Prior to the actual SRS documents there is a vision document which describes the ”vision” of the software. It is a very general description that does not necessarily include any detailed or measurable specifications. The use of SRS is to convert this blurry draft to a detailed, understandable, im-

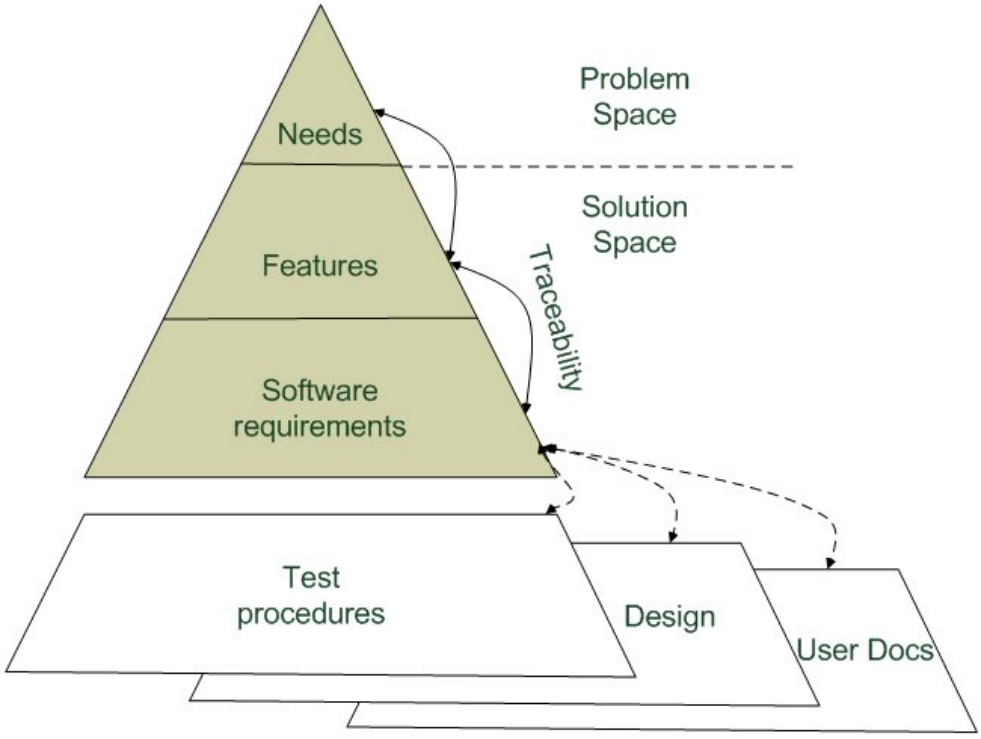

Figure 3.1: Requirements Pyramid. [LW00]

plementable and measurable format. This is represented in the needs section of the pyramid. [LW00]

The whole content of a full SRS package is very complex and can easily exceed a few hundred pages. One suggestion for a set of documents for a SRS is provided by Leffinwell in [LW00]. Due to limitations of the length of this thesis we will examine the following parts for requirements engineering:

The Introduction is already given in chapter 1 and 2, and represents the ”needs” section of the requirements pyramid. As a first step I will identify the Actors that collaborate in using this software. A second step will describe the use cases. These steps are represented in the ”Features” section of the SRS pyramid.

The following steps are part of the ”Software Requirements Section” illustrated in the Requirements Pyramid:

The third step will be to list functional and non-functional Requirements. Those requirements will lead us to step four, The Design Constraints. Due to the scientific approach of this project I am not going to describe the Online User Documentation and Help System Requirements or Purchased Components which are also part of the package, as they do not appear to help solving the research question. As a fifth step I am going to discuss the interfaces of this Software, including a data model, which is

not included in Leffingwells suggestion. The description of Licensing Requirements and Legal or Copyright Notices do not appear to be relevant in that scientifically context of a Thesis. Those steps are commercially relevant but would need to be examined in a separate thesis from a law science point of view. [LW00]

The relationship between requirements engineering and the actual Software Design is shown on the bottom of the pyramid. Here the connection between Test procedures, Design and User documentation is pointed out. It is also important to mention that the pyramid is not a straight top- down approach as there are always feedback processes that can influence earlier steps. If complex needs result in a large number of features that result in complex requirements then the customer might change his actual problem description to allow the development of an easier and cheaper solution. [LW00]

After defining the requirements in this chapter, the next chapter will provide a Data Model for the exchange of information as well as Context-, Behavioral- and Object Models.

In the following subchapters I will first provide a short description of the method of examination before actually examining the subjects.

3.2 Actor Survey, identifying stakeholders

A Stakeholder is "anyone who could be materially affected by the implementation of a new system or application" [LW00]. Leffingwell differentiates between direct users of the system, which are the main target group, and indirect users, which are just affected by the outcome of the system. This affection can take place in various ways, possible indirect stakeholders are controlling government units (especially in healthcare domain) and developers of software that shall be integrated (e.g. Hospital Information Systems). In the context of this thesis it is neither necessary nor possible to identify all indirect stakeholders as it does not describe a real life situation where real companies are involved. It appears sufficient to describe the direct users and analyze their needs.

In our Scenario on the one side there are persons, being typically on route, for business or privately. These people can be private users or professionals wanting to upload information using a mobile device. This is a very diversified group of users. It can consist of old people using their phone for submitting health related information, nurses uploading information for a patient, a journalist uploading a picture or an article including different descriptions, or any kind of other person that wants to submit information to a service provider.

On the other side there is a Service Provider that has an interest in receiving infor-

mation provided by the user. This consists of information from mobile devices, but also from PCs. The providers might use the information for any kind of purpose. Examples for potential service providers are a doctor (for automatic processing in his information system), a publisher (for categorizing pictures and immediately storing them in a database enriched with keywords), or a web 2.0 service provider (for allowing users to directly upload content to their profiles). They might have an interest in receiving data as fast as possible by as many people as possible without requiring the use of a standard PC. The Service Provider is the second user of the System.

The Service Provider must not necessarily be the provider of the middleware that connects his systems to the users' device. There might also be a third Party involved: The Middleware Provider. He has the interest of connecting users with service providers by providing a platform that allows the easy data interchange between those two parties. In our scenario the middleware provider might connect multiple users with multiple service providers. He normally might provide his services with the goal of earning money for the use of his service. He might either charge Service Providers or customers or might follow another business model. Examples are trying to develop a standard for data exchange or have the intent of selling the whole company after it gained a large amount of customers. Another business model would be collecting knowledge about users and providers and using this data for business or commercials. For reaching these goals the middleware provider has to ensure the usability of his software. The middleware provider is not an actual user of the System, but the one running it. Figure 3.2 shows the main stakeholders in that constellation.

Figure 3.2: Main Stakeholders.

3.3 Use-Case Model Survey

Use Cases "describe a sequence of actions a system performs that yields a result of value to a particular actor" [LW00]. In other words they identify the "who", "what"

and ”how” of a systems behavior by pointing out the interaction between a user and the system. This is very important information for designing further details about the system. They are focused on describing what the system does for a user. The use case model describes the totality of the system's functional behavior, including all required use cases [LW00].

UML provides nine diagram types for presenting use cases in different levels of abstraction and with a focus on different properties of those use cases. Some of those diagrams will be presented in later subchapters of this chapter e.g. the Sequence and Collaboration Diagram for showing the required interfaces. [Kec09] Figure 3.3 presents a User with the use case ”Upload Data”.

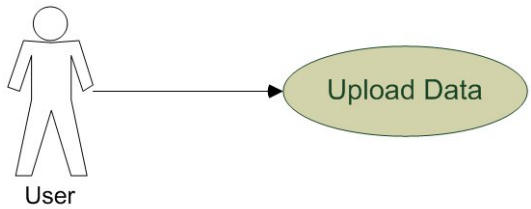

Figure 3.3: Use Case: Upload Data.

This use case is just one of the possible use cases. All possible use cases are presented in Figure 5.3, where use cases are shown, being part of the system to design. The borders of that system are not possible to define yet. There might be different systems like local applications on mobile devices, middleware or adapters to professional systems (e.g. Hospital Management Systems) implemented. For defining the use cases model we see all different applications as one system, which integrates all possible use cases. Figure 3.4 shows the use case model. In this model the above described actors ”User” and ”Service Provider” are involved. First users upload data. The use case ”Upload Data” includes the use cases ”Tag Data” and ”Select Service Provider”, as this system is built for uploading tagged data to a service provider. Afterwards the service provider receives data. This requires that a user has uploaded data.

For a better understanding use cases now are specified in a more detailed way for describing the functional behavior of each use case.

Description of Use Case ”Upload Data”:

1. The User selects the file he wishes to upload. Therefore a screen with a file selection interface is presented. He confirms his selection.

2. Use case ”Select Provider”

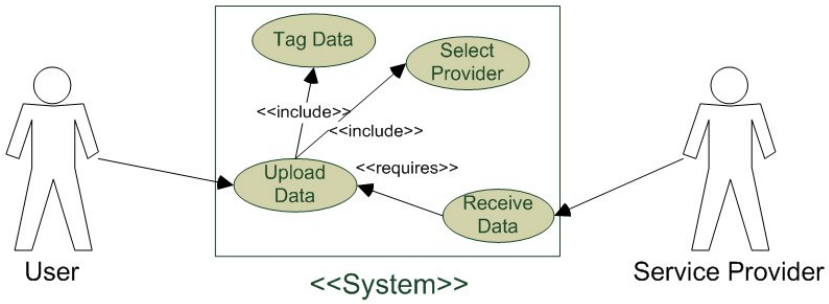

Figure 3.4: Use Case Model.

a) The user sees a screen where he can select the provider he wants to submit the data to.

b) He confirms his selection

3. Use case ”Tag Data”

   a) The user sees an interface where he can add tags. Those tags are predefined.

   b) He can select a tag from a list or find it by typing the tag in an input field.

   c) This process b) can be repeated as often as desired.

   d) He confirms his selection.

4. He sees a screen where he can read his tags and the provider the data is going to be submitted to. He confirms everything by selecting a ”Submit Button”.

This use case is still simple. It just allows the user to add information by selecting one tag. As discussed in chapter 2 an ontology is a typed relationship between concepts defined as ”an explicit, formal specification of a shared conceptualization of a domain of interest” [Ehr07]. Later we discussed the differences between associative, mono-, multi-hierarchical and typed relations. The main difference was that typed relationships contain more pieces of information than the others. They include the type, concepts are connected. This allows the developer of typed relations to include more information than in the other relationships. Using OWL or RDF and RDFS it is possible to define classes and create elements being members of those classes. An example would be the class ”car” defined in RDFS with a property owner. Those elements would need a unique identifier. This identifier could be ”international license plate”. That class could have many different properties as well, like ”horse powers”, ”convertible” or ”brand”. Those properties could include different types of properties: ”horsepowers” as integer, ”convertible” as boolean and ”brand” as string.

In RDF there could be many car elements defined like the ”finish Saab convertible with 156 HP ”FI - OHV 331””, the ”German Skoda with 90 HP ”D - WI - ND 546”” or the ”Luxemburgish Ford Fiesta ”L - OW 22052” with 54 HP”. Those cars could have an owner who would be a person. Therefore we would define the class person in RDFS and could create many objects of the type person afterwards. Then we would have to define the relationship ”owns” in RDFS, for allowing the car to have an owner.

This example shows that using ontologies for tagging can be much more complex than just assigning a simple word as a tag:

First it is possible to assign a simple tag, like ”FI - OHV 331”. This would enable the user to assign a Tag that is computable by the recipients system. In case of an insurance company, the user could for example assign the tag of his car to a photo taken after an accident.

Second it might be possible the user wants to submit a document considering a person. Therefore it might be useful to find out what persons are known to the system. So the user would first select the concept he wants to use ”person”. Than the system should provide assistance in showing all persons that are known to the system. He further could search for persons having the property ”lives_in” with the value ”Tahiti”.

Third it might be possible the user wants to enter a tag that is not known to the system, for example a new car. RDFS allows it to define mandatory and optional properties. So the system could present the user an input-form where he could enter all required and optional information. He might also want to create a new relationship ”owns” for telling the system who owns the car. Afterwards he could use that newly created object for tagging.

Fourth it could also be possible that the user wants to create a new concept for example ”motorbike”, which might be very similar to the concept ”car” to the system. Therefore the system would have to show a list of existing concepts and allow the user to select one as parent concept for the new concept to create. Then the user would have to be able to add, change or remove properties of that parent concept for defining his new concept.

Scenario one and two don’t involve the change of the ontology itself. They just present the existing ontology to the user and allow him to navigate through it. One of the big advantages of tagging using ontologies is that the user is able to navigate through the objects available for tagging. This definitely has to be supported by the system.

Scenario three and four on the other hand involve a change of the existing ontology. Using RDF scenario three would allow the user to change the RDF file by adding new

objects, like a new car for example. Scenario four would go one step further and allow the user to also change the RDFS file and so the class structure of the system. The advantage of allowing that user interaction would be that the user is free in his choice of tags. The disadvantage would be that there might be not properly defined concepts and compatibility issues. In the extreme case of a standardized RDFS that is used by many insurance companies all over the world it is definitely not desirable that a normal user is able to change the scheme.

The change of the RDF File itself, described in scenario three, might be acceptable e.g. by introducing a new car as a new object. In the case of a car it would be necessary to connect it to an existing person, which is in some way known to the insurance company. A possible scenario would be the picture of the opponents' car in an accident that is not insured at the same insurance.

In the scientific scope of this project, it is at least necessary to discuss the possibilities for this extended tagging mechanism. This will be done in the next chapter.

The use case "Receive Data" can be described the following way:

1. This use case gets initiated by the system.
2. A message gets delivered to the service providers system, including the new uploaded data as well as user information and tags.
3. The provider's system processes the data and decides based on roules what to do with it.

Both use cases have just one actor. Never the less the second use case relates to the first use case in the way that it gets initiated by the first use case. Use case II requires use case I.

After defining the use cases, the next subchapter will define the requirements.

3.4 Requirements

The vision of the software led us to the use cases that described the user interaction with the system. These use cases show what the system is supposed to do. Based on this information we are able to phrase the requirements.

Software requirements are defined as "a software capability needed by the user to solve a problem to achieve an objective" [LW00] or; "Requirements are about the phenomena of the application domain, not about the machine. To describe them exactly, we describe the required relationships among the phenomena of the problem

context." [Jac95]. This second definition shows that requirements do not illustrate the processes needed for reaching a goal, but the supporting elements and parts of reaching it. Requirements should not contain any implementation, testing or project management details. These details should be integrated in the design process. [LW00]

There is a discussion between developers on the topic how detailed the requirement list should be [LW00]. For the purpose of this thesis just the main points will be formulated. Secondary information, e.g. "when a user tries to log on to the system and enters a wrong password, he shall be redirected to a screen, where he is asked to reenter his password again." is not helpful for understanding the main requirements. This subchapter is divided into functional and non functional requirements.

3.4.1 Functional Requirements

Functional Requirements express the behavior of a system. They are mostly stated in phrases like "when User does A, System will do B" [LW00]. In this section the author presents the functional requirements of the software to develop:

1. When User starts software, he has to log on using username and password. If it his first logon, he has to provide user details and select a service provider.
2. After logging in, a selection screen is presented where the user selects a file to upload.
3. When User selects the file to upload, the data is stored and a provider selection screen is presented.
4. When User selects a provider, the selected provider is stored and a tag selection screen is presented, including the supported tags.
5. When User selects a tag, the tag is stored and a screen is presented where he is asked if he wants to add more tags.
    a) If user wants to add more tags, a tag selection screen is presented again.
    b) If user wants to finish tagging, a control screen is presented, where the data, provider ant tags are shown.
        i. If he agrees to the collected information, the data is submitted to the provider, including personal information and tags.
        ii. If he does not agree, he can change the data on a change Screen. Back to step 2, 3 or 4b

6. The data is transferred to the provider, including the tags.

7. If a user sent a message to a provider and the provider receives the information the user sees a confirmation of the submission.

Those requirements are described using a flow of events that can also be represented in a UML activity diagram for a better visualization. Figure 3.5 illustrates the flow of events in an activity Diagram.

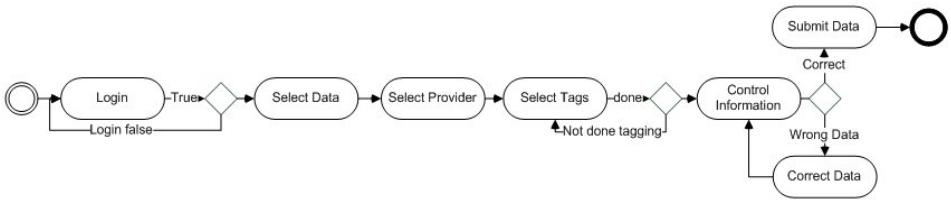

Figure 3.5: UML Activity Diagram.

3.4.2 5.4.2. Non Functional Requirements

Functional requirements describe what a system is supposed to do. Non-functional requirements on the other hand describe some attributes of the system or its interaction with the environment. Those can be divided into usability, reliability, performance and supportability. [LW00]

Usability can be expressed using properties like estimated training time for users to learn operating the system, the time a task requires or the availability of help systems. Characteristics describing **reliability** are availability (e.g. 99.9%), mean time between failures, maximum bugs and accuracy. **Performance** can be described using response time for a transaction, capacity (number of customers the system can accommodate) or throughput (transactions /second). **Supportability** describes the ability of a system to be easily modified. This includes as well enhancements as repairs. Enhancements can be any kind of changes the business deems necessary. As it is difficult to project changes that are necessary in the future, it is also difficult to measure supportability. It is possible to require a modular architecture that should be implemented in modern programming on principle, the use of special managed database environments or special programming languages that allow quick changes. In the following part I will discuss the non functional requirements for our software. [LW00]

Usability:

- The System has to be usable by any user, without a long training.

- The input methods of the mobile device have to be supported.
- The input structure has to be clear and easy to understand, even on a small screen. The user shall be guided by a wizard.
- The user has to see where he is in the process all the time.

Reliability:

- A 100% reliability is not required.
- It is desirable that the service is highly available.
- The security and integrity of submitted information has to be ensured, as there can be personal or otherwise trustworthy information involved that gets sent through communication channels.
- The system should report errors as they accrue. If it approves the delivery, this has to be reliable.
- Bugs should be minimized.
- Bugs should be fixable quickly.

Performance:

- As the amount of users is not predictable, the backend of the system should support enough users with an appropriate response time.
- The system on the phone should allow a quick response time.
- The download of the required information to the mobile device (tags) should not take longer than 5 seconds.
- The upload of the data should not take too long, but as the data can have any formats only networks limitations for speed are relevant and not changeable.

Supportability:

- The system should be easily modifiable to support new mobile devices.
- As well it should be easy to integrate new service providers.
- New document types should be supported, even if not known yet.

Some of these non-functional requirements refer to the special circumstances of mobile devices like bad network coverage, different operation systems, different browsers, different input types and varying screens. It is definite, that not all devices provide the power to fit the needs of the system to develop.

3.5 Design Constarints

Design Constraints impose limitations on the design of the system that should be built. They can be defined as "restrictions on the design of a system, or the process by which a system is developed, that do not affect the external behavior of the system but that must be fulfilled to meet technical, business or contractual obligations" [LW00]. Examples are the use of a concrete programming language, not to use a specific database or regulations by laws, government organizations or business standards. Normally the technical design of software should be the choice of the programmers, design constrains limit that choice. Finally they have to be accepted as if they were technical requirements. [LW00]

There are some regulations applicable to our system, depending on the data that should be submitted. German law gives a lot of limitations for the use of medical data in "Bundes Datenschutzgesetz", the "federal law of data privacy protection". For the transmissions of medical data it requires confidentiality, authenticity, integrity, availability, reversibility, validity, legal certainty, non deniability of transmission and a usage commitment. [BWB+02]

There are similar regulations for journalists, or confidentiality agreements between companies and their personal. For the scope of this thesis I will ignore these design constraints as they are, considering the amount of different domains this system could be used in, too complex for an integration to the software.

3.6 Interfaces

This section provides requirements considering the interfaces of the software to develop. Interfaces are the part of a software that connect to the software's surrounding. Leffingwell's suggestion for the SRS Package suggests the diversification of User Interfaces, Hardware Interfaces, Software Interfaces or Communications Interfaces. [LW00] Interfaces can also exist between classes, subsystems or components. Here they specify a set of operations executed by one part of the system. They also include the number and types of parameters as well as the type returned. [Kru98] Those Interfaces are not

part of requirements Engineering but of System Design so they are not part of this chapter.

3.6.1 User Interface

3.6.1.1 General Requirements

Bjarne Stronstrup, the originator of C++ said: "I have always wished that my computer would be as easy to use as my telephone. My wish has come true. I no longer know how to use my telephone." [Pre05]. This quotation shows the importance of a good user interface design.

Our software requires just one user interface. It is located on the mobile device and allows the user to upload and tag data. The service provider receives the submitted data in a format that integrates to his system. The development of a user interface is not required for that.

Pressmann recommends some principals for a user interface design. The user should only do what he has to do. Unnecessary operations should not be demanded. The user should be flexible in the way he interacts with the system. On the other hand guidance through the process is required. He should at any point be able to undo his operations. The design should be consistent. "Things that look different, should act different. Things that look the same should act the same." [Larry Marine, after [Pre05]]. He should at any point see the main buttons for navigation and for correcting inputs. Those buttons shall stay at the same point of the screen and not move. [Pre05]

As presented in chapter 2, Considering the limited space on the screen of a mobile device, it is necessary to create a consistent input form. There should not be too many things happening at once. It is better to have three steps the user can overlook and understand in seconds than having one step on a screen that make the user scroll and maybe lose his orientation in the process. A constant status display, presenting the current step of the process, also supports a good orientation of the user.

The user interface has to provide adequate ways for input using the devices standards as well as an optimal output considering the screens specifications of the device.

3.6.1.2 Ontology specific Requirements

The user interface should support the assignment of tags, known to the system, to browse those tags by concepts and properties for allowing the user to select an appropriate tag and to add new objects to existing ontologies, showing mandatory and

optional properties. This browsing for concepts should be implemented in multiple steps for allowing the user to find the appropriate tag in an easy way.

3.6.2 Hardware Interfaces

Hardware interfaces describe the ways of connecting to required hardware. [LW00] In this scenario there is no specific hardware required but the mobile device, and the servers. This hardware is usually fully supported by the implementation environment, as it was especially developped for the devices.

3.6.3 Software Interfaces

Software interfaces describe the required communication processes to other parts of the system that are not part of the SRS Package. [LW00] Here all parts of the system are part of the package, so no software interfaces are required. The communication to the Service Provider is presented in the next subchapter.

3.6.4 Communications Interfaces

In this part of the thesis the system is still seen as everything between the mobile device and the Service Provider. The development of other middleware will be discussed in the next chapter. The communication interfaces describe the communication to other systems, not between parts of the same system [LW00]. This section describes the Networks and Protocols to be used for connecting to other systems. A Context Model is a frequently used model for showing the surrounding systems. [Kec09] It is used for defining the system boundaries. Figure 3.6 shows the context of our system.

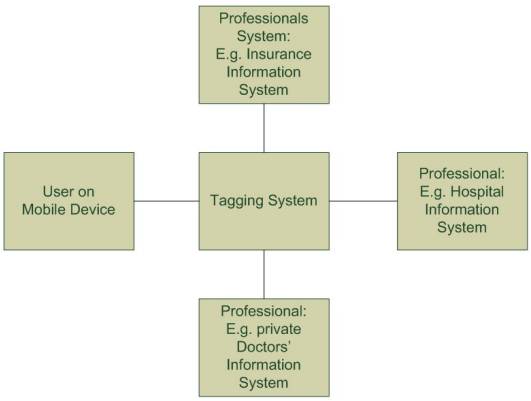

Figure 3.6: Context Model of the Tagging System.

There are two different systems in the context of our system. The first one is the users small device, the second system is a professionals system. For taking a further look on what kind of communication is needed, a Data-Flow Diagram is recommended. A Data-Flow Diagram illustrates the data-flow of the system to build and shows the points of dataflow where external input is required. [LW00]

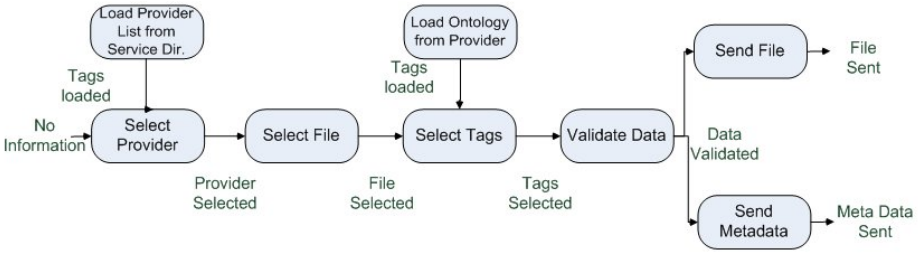

Figure 3.7: Data-Flow Diagram.

Figure 3.7 shows the Data Flow Diagram of our System. It demonstrates that there are four necessary connections to the surrounding Systems:

At first the connection is established by loading the provider list from a service directory, which might also be included in the system. Secondly the ontology is loaded from the selected providers ontology service for being able to display the available tags, concepts and relationships between those to the user. Thirdly the selected file is sent to the provider and fourthly the selected metadata as well as the user information stored in the system is submitted. It will be necessary to design interfaces for these four connections.

The different connections need to be established using the internet. The standard protocols are TCP and IP. Considering privacy issues, which are relevant in many domains, especially the health care domain, it is necessary to provide a secure connection for submitting the information. This can be realized by encrypting the data before transmission or by using safe communication channels like SSL. The SSL protocol is also available for modern mobile devices using their wireless communication features [e.g. [msd08]]. [Bur04]

During the development process the protocols for communication will have to be specified, including the applicable standards, which will be discussed in the next subchapter.

3.7 Applicable Standards

In chapter 2 I presented different languages for representing ontologies. RDF with RDF Scheme and OWL, two XML based standards, were introduced and discussed. As both standards are in use, it is necessary that the system includes support for both standards.

One problem considering OWL and RDF is the file size of large ontologies. A large OWL file can easily reach many Mega Bytes. This is a serious problem, considering those onwologies have to be transferred to the mobile device. As the ontologies change regularly, the user might need to update the file before the transmission of data. This is not an acceptable solution for a mobile surrounding. On the other hand it is not practical for the service provider to implement smaller ontologies especially for the mobile customers. This problem leads us to a design question, how to transfer only required pieces of information. This question is already part of the implementation details, which will be discussed in the next chapter.

The uploaded data, which gets sent from the user to the provider, should be submitted in the original format. For ensuring compatibility, the service provider needs to define what data types are supported. A check, whether the format of the file to upload is accepted, has to be performed by the system before upload.

There has to be a standard developed, that contains the information, associated to the file. XML is a very general standard that allows the definition of an XML Scheme, where all required information can be stored. Chapter 2 describes details about XML and XML Scheme.

4 Design of an Implementation for a Mobile Device Application

After expressing the requirements for the application that supports the goal of allowing the use of ontologies for tagging data in a mobile context, this chapter will give a formal design of an application that implements this functionality. Therefore I will first discuss the system's architecture in subchapter 1. This discussion is relevant for the further design of the application as it determines the necessary communication between the different parts of the system. In the next subchapter I will give an introduction to services and identify some necessary services between the system and the service providers. Subchapter 3 will describe the client server communication required for a shared system that will be discussed in subchapter 1. Subchapter 4 presents a general interaction model and presents an overview of all required communication.

In Subchapter 5 required classes for the server side of the system will be discussed, in subchapter 6 the same discussion will be done for the client side. Subchapter 7 will give some implementation details for the server side functions and finally subchapter 8 discusses the user interface design.

User help modules or other assisting parts of the implementation will not be discussed, as the available space of this thesis is limited.

4.1 General architectural Discussion

As described in chapter 3, there are different parts of the application that run on different machines at different institutions or at different users. Examples are the service directory, the clients system and the professionals systems. As described in chapter 2 there are different mobile devices with different operating systems, screens, input devices etc. This spread setting of our system limits the general organization of the system architecture to models that support multiple locations of a system.

Sommerville [Som04] presents many ways of organizing a system. The relevant system organizations for distributed systems are presented in this subchapter.

The **Repository Model** is a way of organizing systems, where all shared data is held in a central database which can be accessed by all sub systems. The different subsystems additionally maintain their own databases and interchange data by sending messages to other sub systems. [Som04]

Sommerville mentions as advantages of the repository model as an easy way of sharing large amounts of data. Subsystems do not have to integrate connections to other subsystems and the centralization of backups and access control also makes administrational tasks easier. The main disadvantage is that all subsystems have to agree on a standardized repository data model. That means that all subsystems, that are supposed to access the stored information, have to support the standards defined by the repository. For our model that would mean that the data of all participating entities would be stored in a central place, no matter if the user wants to submit information concerning his car accident to an insurance company or information considering his current blood pressure to his private doctor. This architectural approach seems to be not useful due to the different requirements of the different actors. [Som04]

The **Client-Server Model** includes a set of servers that offer services to a set of clients. The servers offer those services to other parts of the subsystem and the clients can call those offered services. There can be many instances of the same client software trying to access the server simultaneously. For the clients it is necessary to know the URI of the servers as well as their abilities. Considering extensibility it is easy to allow new clients accessing one server and it is also possible to add new servers with new functionality. [Som04]

The Client-Server Model seems to be appropriate for connecting clients to one server. It will be further discussed later in this chapter.

The **distributed objects architecture** is another scenario that extends the Client-Server Architecture. Here no distinction between clients and servers is made. All objects can act as clients and servers and are connected using a software bus. This model seems to be oversized for the envisioned software as it would allow connecting any server to any client. Most of participating entities in our model will be clients and those clients do not need the ability of offering services. Due to the limited resources on a mobile device this architecture seems to produce a lot of overload. [Som04]

Wuest describes the **Peer to Peer Model**, which has the principle of equal pears that can all start a communication process and forward communication packets [Wue05]. As our architecture is intended to submit information from one direction to the other only, I will not further discuss Peer to Peer architecture.

Finally Sommerville describes the concept of web services, already described in chap-

ter 2. As mentioned there, services offer their abilities to clients. Those clients can request information, a computation or can send information. As web services offer an easy management through a Service Directory, and so allow the easy discovery of services and communication to various clients and servers, they seem to be interesting for the use of our application. [Som04]

There are many ways of implementing web services in our scenario. The Data Flow Diagram in Figure 3.6 presented different points in the Data Flow, were information from external sources was required or was sent to external services. These points are shown in Figure 4.1.

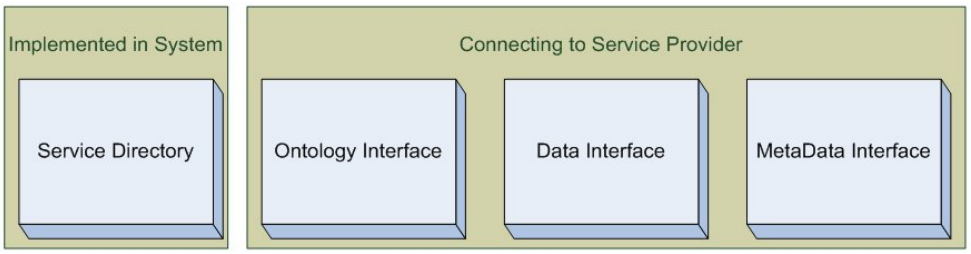

Figure 4.1: Interfaces.

The service directory is responsible for listing all potential recipients of the data. It has to submit a list of those recipients including a description of the services. It has to allow the services to register at the system. [Erl05]

The service directory can be part of our system, but as there are no special requirements for the use in our mobile context and that topic is already widely described in various semantic web and SOA [e.g. [Erl05]] publications I will just briefly describe the functionality of a service directory and recommend [Erl05] for further reading on this subject.

The Ontology Interface, the Data Interface and the Metadata Interface connect to the service providers system. The service providers have to register at the service directory before.

The **Ontology Interface** provides the ontology used by the external system. Thisontology gets submitted in a RDF/ RDFS or OWL file. As ontologies can change over time it is useful to update this ontology file for each tagging operation. In case the user updates the ontology by entering new objects, the ontology interface has to support sending the changed ontology back to the provider. The request and update of an ontology can be implemented as a web service on the service providers side.

The **Data Interface** and the **Metadata Interface** have to submit the data as well as the produced metadata to a service provider. This can also be implemented using

a web service on the service providers side.

Figure 4.2 illustrates the connection of those four different systems to our system and presents the possible connections.

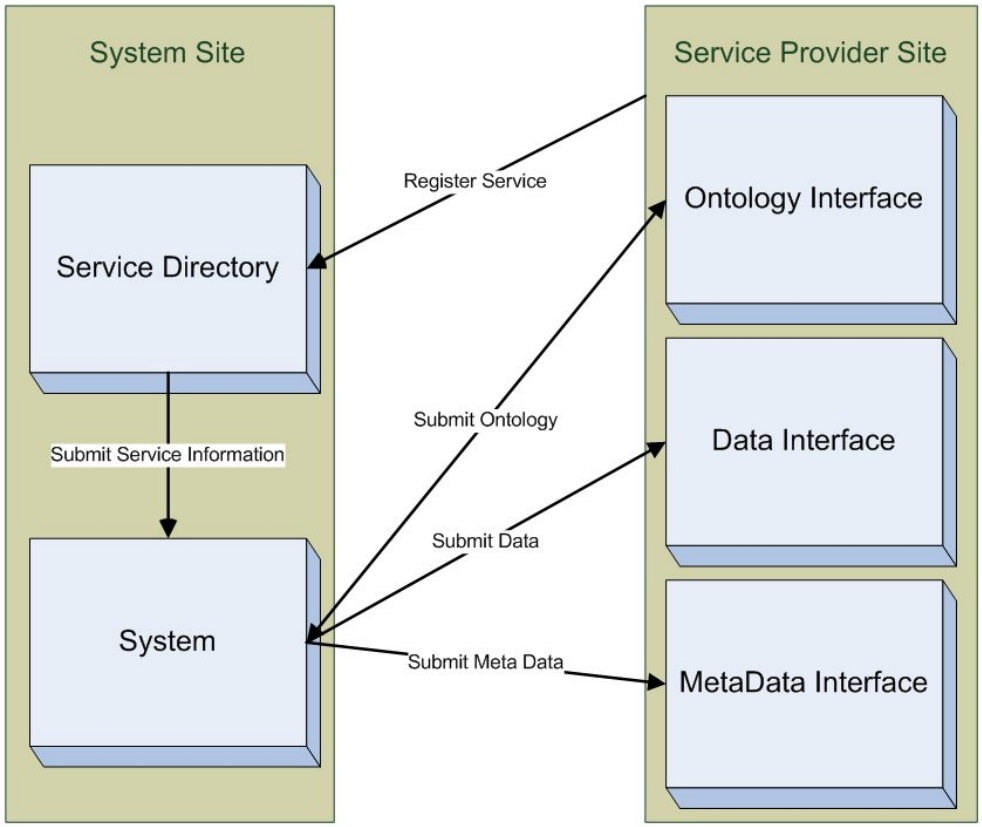

Figure 4.2: Services and Connections.

The following interfaces are discovered:

- Interface I, connection from Service Directory to System.
- Interface II, connection from Ontology Interface to System.
- Interface III, connection from System to Data Interface.
- Interface IV, connection from System to Metadata Interface.

Considering our system is supposed to run on a mobile device it is necessary to discuss the amount of data transported through the different interfaces.

Through **Interface I** the user needs to receive first the names and short description of the existing services. This data is necessary to be received on the mobile device as the user needs a list of all available services, for selecting the one he wants to upload his

data to. It would also be possible to split this data into categories or first submitting the frequently used services. After selecting a service provider in a second step, the users system needs to receive further information on the service, including the URI, accepted file formats and registration information. It is not necessary to receive the information of step 2 for all services. Therefore it can be useful to split the information provided by the service directory into small parts.

Interface II - submitting the ontology used for tagging to the system - may transport big amounts of data. Large ontologies can easily reach many Mega Bytes. This information might be split and sent in small pieces, for allowing shorter load times. A further discussion on the split of data will take place later in that chapter.

The Data submitted through **Interface III** is directly coming from the mobile device, as it is the data that has to be submitted, so there is no other way than sending it straight from the device.

The metadata submitted through **Interface IV** are tags that get selected on the mobile device as well as some user information that might also be stored on the device. As users need to input the tags on their device, the amount of data produced there might not be too large.

Splitting the data for Interface I and II into small parts, might either be implemented straight in the service - the service directory and the systems service providers - or as well using a central server, that connects to the different service providers and forwards parts of the information as required to the mobile device.

For making the connection to the different service providers as simple as possible, the second way of programming an intermediate system seems to be preferable. This scenario is presented in figure 4.3.

This architecture allows us to define standard web services, provided by service providers. Those services providers have to register at the service directory, and provide their services for publishing their ontologies and receiving the data and metadata. Those services will be described in subchapter 4.2, including a more detailed introduction to service directories.

Another way of presenting the systems' architecture is the use of an Deployment Diagram. Deployment diagrams are used for modeling the architecture of a spread system during the runtime. Therefore they specify the hardware and software environment of the different parts of the system. [Kec09] Figure 4.4 illustrates the deployment diagram for the system. Here mobile devices are one entity that connect to the intermediary system, which is a web server that is located somewhere in the internet. The mobile devices need to be able to connect to the internet and allow the execution of

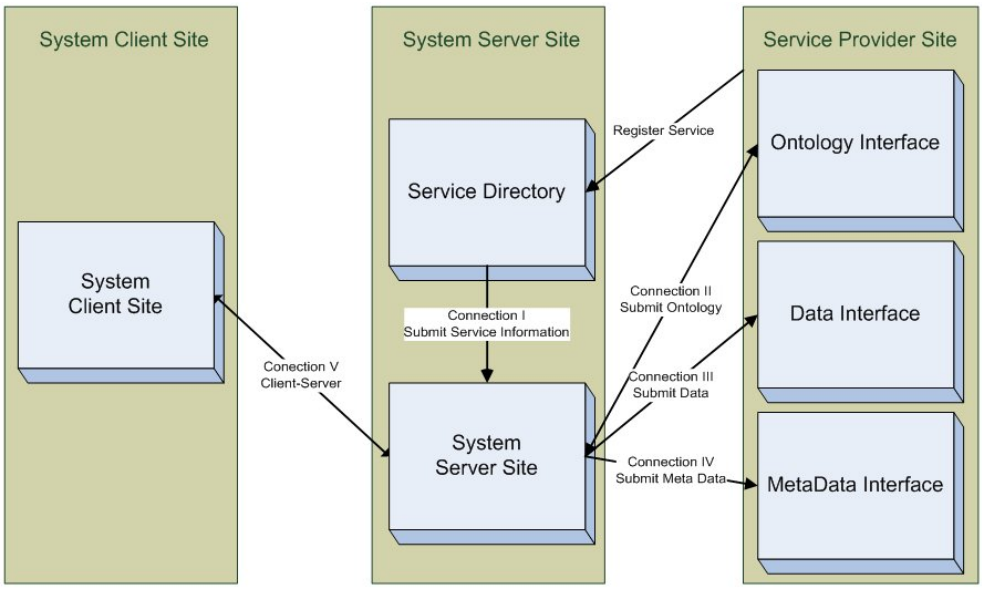

Figure 4.3: System Architecture.

new software, either in a virtual machine, a web browser or natively. The web server needs to provide an execution environment that suits the needs of the development platform the application is developed in. The Service Provider provides a web service that connects the service interface to the professionals system on the back end. The service directory consists of a module for registering services that communicates with service providers and a module for service publication that communicates with the server side of our system.

Some parts of this deployment diagram, e.g. the use of services, will be discussed later in this chapter. The service directory and the connections between the server side of the system and the service providers will be briefly described in the following subchapter. Afterwards I will introduce the client server architecture, connecting the mobile device to the system server side.

4.2 External Service Connections

We identified the necessary connections between the system and the surrounding services in chapter 4.1 as a part of the general architectural discussion. Those connections will now be described more precisely as services. First I will briefly describe the functionality of a service directory. Afterwards the services, the system's server side is connecting to, will be specified.

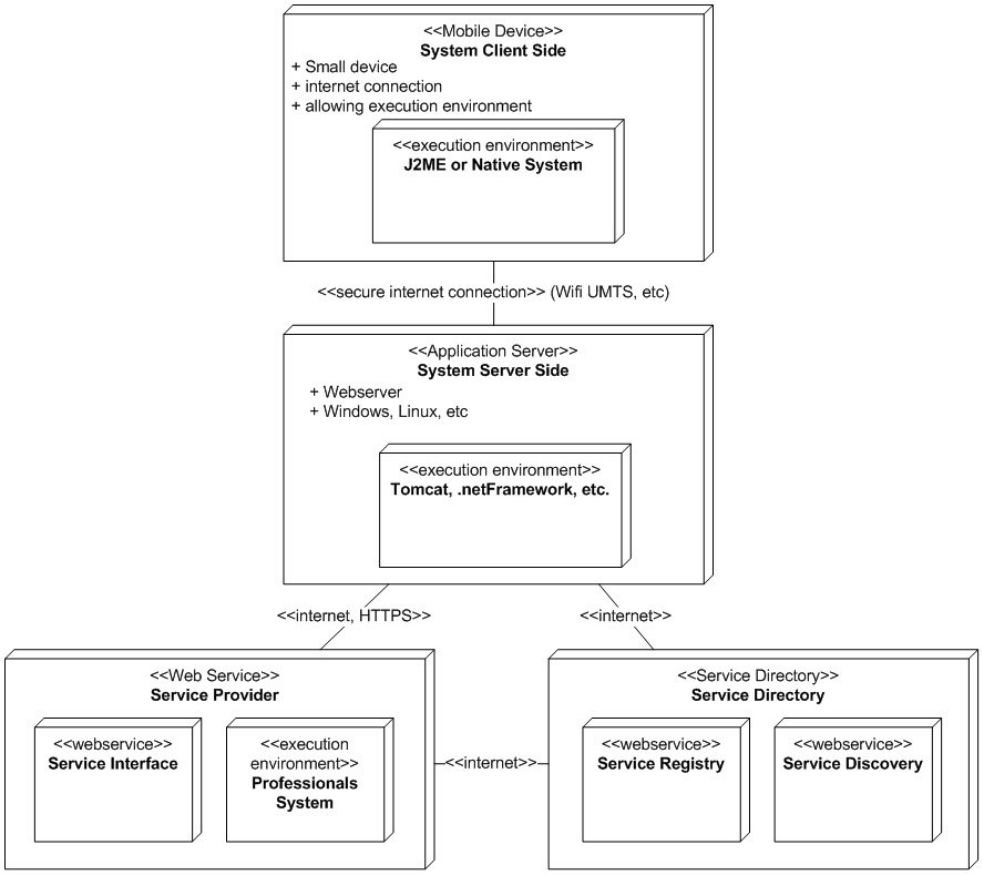

Figure 4.4: UML Deployment Diagram.

4.2.1 Servie Directory

The following subchapter will explain the functionality of a service directory and will present a component diagram. Therefore I will discuss the basic technologies for a service directory and define some standards.

UDDI is a standard for Universal Description, Discovery and Integration of web services, that is widely accepted in literature [e.g. [ACK04], [Erl05]]. UDDI specifies APIs for browsing and searching for business services, a data model for describing those services, nodes and registries. The Service Registry Protocol describes standards for the visibility and reuse of service components. This includes dynamic location, binding and discovery of services. Figure 4.5 shows how a UDDI registry can be used. [OAS05]

The UDDI Registry is the central point in that architecture. It contains links to service descriptions, where services are described using WSDL, the web service description language. The service description contains all required information about the service. The different parts of a service description are shortly defined in the following [ACK04]:

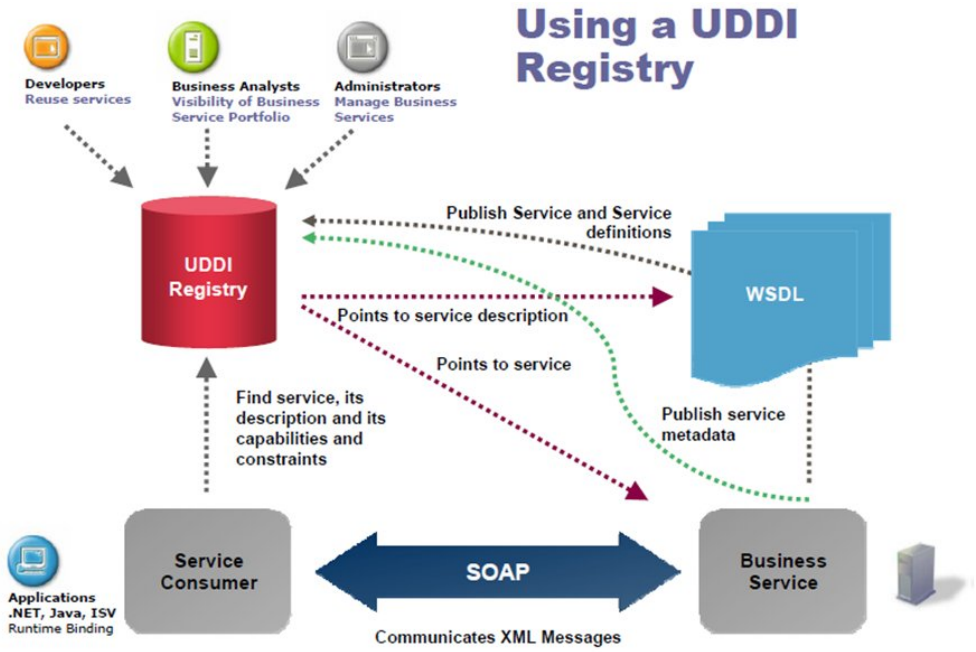

Figure 4.5: UDDI Registry. [OAS05]

The **Common Base Language** defines the language all communication is based upon. XML is a widely accepted, flexible base language.

The **Interfaces** describe the way of addressing the service. For example the URI and the transport protocol are defined here.

Business Protocols define the way services handle requests. For example the required steps of an interaction are defined in that part. E.g. in an "'order-Service"' it is first necessary to select the items before the paying process can be started. In the case of our application, Interface III and IV, described in Figure 4.3, can be combined to one web service where first the data and second the metadata is submitted. This order of events would follow a defined business protocol. [ACK04]

With the above described information (Common Base Language, Interfaces and Business Protocols) it is possible to detect and afterwards use a web service. This information has to be provided by all service providers and submitted to the service directory. Afterwards the Services can be used by our system. Figure 4.6 illustrates an UML component diagram for the service directory. It provides a socket "RegisterService" that allows service providers to register their services and an interface "GetServiceList", that returns all required Information about the different services. The implementation of the socket and interface are out of the scope of that thesis, their existence is never the less important to point out for understanding the whole

process.

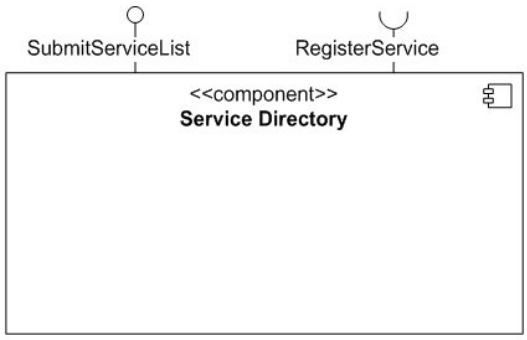

Figure 4.6: UML Component Diagram Service Directory.

4.2.2 Services the System is connecting to

We identified four services necessary for the communication. Connection I connected to the service directory and was already described. Connection II connects to the ontology service. The ontology service has to provide the ontology for tagging and has to be able to receive new concepts if the user wants to add new information. As described in the previous subchapter, Connection III and IV can be combined to one service that is responsible for receiving the tagged information as well as the attached metadata. It is further possible to combine this functionality with the functionality of adding new elements to the ontology. So there are three different things the service needs to provide that have to be described in the WSDL file:

First the file, describing the service, needs to contain the common base language which is XML. It needs to contain also information about the connection standards which should be for securing the connection channel. The service description needs to contain as well the description of the interfaces, which means the functions that are accessible for clients. The URI of the service is also required, as well as the URI of the ontology that shall be used. Necessary information is as well the format that is used for storing the ontologies. Here OWL or RDF/ RDFS are possible.

There are two interfaces required: The first interface is a combination of connection III and IV. It receives the information uploaded by the user, the attached Metadata and, as third parameter, additional information that might be added to the ontology and was entered by the user. This Interface requires the uploaded data in the original format, the attached metadata and, if the user created new concepts, the new concepts in an OWL or RDF file. Calling this service could happen using the two routines:

SubmitInformation(Data, MetaData, userinfo) returns bool for confirmation.

SubmitInformation(Data, MetaData, userinfo, OWL/RDF File) returns bool for confirmation.

This interface is an overloaded interface, that can be called using an attached OWL/ RDF File or not.

The detailed format for the XML file containing the metadata will be discussed later. When the service has received the information it has to be processed by the recipients system and the service has to confirm the delivery. The actual processing of the information is not part of that thesis and can vary a lot considering the different possible recipients.

Those two interfaces are enough for managing the required tasks. The interaction of these services is presented in Figure 4.7.

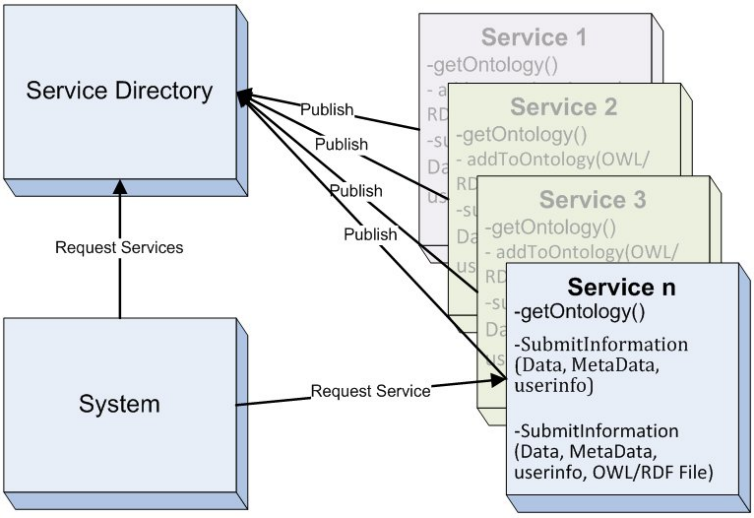

Figure 4.7: Service Interaction.

All services are published to the service directory. Our system requests a list of all services registered to this directory. Afterwards it requests the exact descriptions of the service the user has selected.

The data model for submitting the metadata to the service provider is presented in Figure 4.8. It is necessary to submit technical data, user information and the selected tags.

The "'technical data"' section contains information about the date, the data was submitted, the ontology that was used, the service the data was submitted to, the name of the submitted document and the document format. All information except the document format is mandatory, as there might be files without document format

information.

The "'User Data"' section is either a user name which has to be stored in the system for a specific service or some information on the user like his name, birthday or address. This information does not need to be mandatory for all information as there are services possible, where the user does not want to submit his detailed information. For professional services in the health care or business context this information might be mandatory. A service might require a username for this context. Finally it is necessary to submit the tags selected by the user. These have to be submitted using the unique identifier provided by the ontology.

Mandatory information can be defined by each service provider, by including a XML Scheme to the WSDL file, that defines some values as mandatory, some as not.

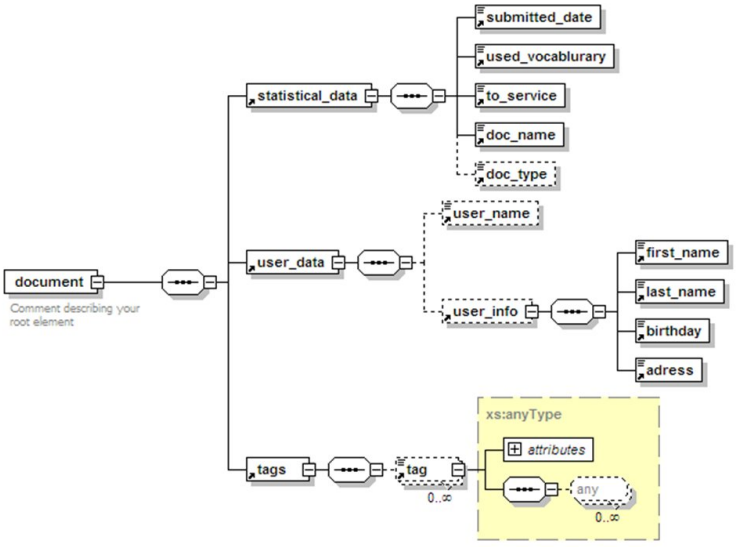

Figure 4.8: Data Format for submitting Metadata.

4.3 Client-Server Conections

After describing the services and the connection of the server side of the system to the services providers the following subchapter will discuss the connections necessary between the client and the server side of the system, introduced in Figure 4.3 and 4.4. As this connection can be realized using different ways, I will first point out the required functions and second discuss architectural models for connecting clients and servers. Third I will provide more information on web services, as they will be the chosen architecture for connecting client- and server side of the system. Afterwards I

will list all required services, and finally connect them using an UML Sequence diagram for illustrating the interactions.

4.3.1 Required Server Side Functions

The server side of our system has to submit the relevant information to the web services and also has to forward the received data from the web services to the clients' device. This information is - from Server to Client - first the available services and second the ontology, provided by the service. The information to be submitted from Client to Server is first new Elements, that shall be added to the ontology, second the data that shall be submitted to the service and third the attached metadata. There also has to be a possibility to change the user information on the small device. These are the functions that require communication between client (mobile device) and server (server side of the system). This can be realized by providing interfaces on the server side that can be called by the client. Those functions on the server side will be described in this section.

GetAvailableServices() This function submits the available Services from the server side to the client side. It returns the available services in an XML File. It is enough to submit the service name, description and an identifier.

GetTags() This function has to manage the available tags. As discussed before, if the ontology is large, it might be a lot of tags. Therefore it is necessary to submit just a small selection of possible tags to the small device. The amount of data that has to be submitted in a special situation is limited by the size of the screen. If the user is not able to see more than 10 concepts on his screen, there is no need to submit more than 10 concepts at once to the mobile device. The data for the next view can change on a keystroke of the user. If the user for example wants to browse concepts the first concepts should be loaded in an alphabetical order. If he presses the key "'G"' all concepts starting with the letter "'G"' should be submitted. If he presses "'H"' afterwards, all concepts starting with "'GH"' have to be submitted.

On the server side of our system this function has to parse the ontology and generate a list of tags, which can be Objects, Classes or Properties. It also has to generate a list for each concept, all objects and a list of properties and their values. It is for example also possible that the user wants to see all objects, having the property "'is a"' with the value "'doctor"' and have as well the property "'located in"' with the value "'Helsinki"'.

As this function has a lot of different functionalities it is useful to divide it further into different functions:

First the user might search for a Concept (Class). Therefore he could use a function GetConcepts(). After searching for a concept the user might search for available properties for that concept, using the function GetProperties(). After doing that, the user will have to select a value for the selected properties so the function GetValuesOfProperty() has to provide the available values. If the user finally searches for objects, the function GetObjects() has to be called with arguments, if the search is limited to objects having a property with a given value. If the user wants to get any available Tags neither limited to concepts or properties, the function GetThing() can return any available objects. These possible functions are illustrated in Figure 4.9.

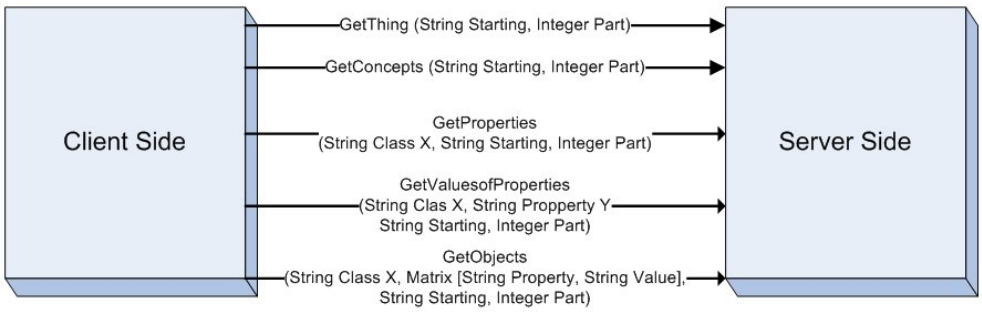

Figure 4.9: Ontology specific Functions.

The format for all parameters should be string, except for the property -value tuples. Here it seems appropriate to submit the Property - Value pairs in a 2 x n Matrix. This Matrix could be easily browsed with a foreach statement, processing all property-value pairs and would not limit the amount of desired properties.

The string "'Starting"' in each function stands for the typed first letters of the tag. The integer "'Part"' stands for the position of the first tag that shall be submitted in the result list. As an example for one of those functions, requesting anything located in Sweden could be realized by calling the function

GetObjects(Thing, [located_in, Sweden], NIL, 1)

for receiving the first part of that list, without a starting letter given.

I will now describe the other functions the server side has to provide for the client side.

SubmitData(Data)

This function submits the file that has to be uploaded to the server side of the system. There it gets stored for the submission to the service.

SubmitMetadata(XML File)

This function submits the selected tags from client to server.

GetUserInfo(username)

This function returns an XML File containing the user information containing name and address.

SetUserInfo(userinfo.xml)

This function submits the user information from client to server. On the server side it updates the user information.

RecieveRequiredInformation(Concept)

This function sends the required information for a new element for the ontology to the client system. The server has to read the OWL/RDFS file for detecting the mandatory and required elements of a concept and has to submit this information in a new OWL file containing the definition of the selected concept.

SubmitNewElement(OWL)

This function submits the new element in an OWL file containing the new information that shall be added to the ontology.

This services show, that most of the logic is provided by the server side of the system. This has to be supported by an architecture, that allows the client server communication. As described in the previous subchatpter, there are different architectures possible for realizing this communication. It could be Web Services, provided by the Server side or a Client Server Architecture.

4.3.2 Architectural Thoughts on Client Server Connections

After stating the required functions on the server side in the previous subchapter, I will now discuss some types of architecture for implementing a client-server architecture. Sommerville [Som04] describes three different layers in Client Server Applications: He first presents the ”Presentation Layer”, which is normally located on client side and has to present the information processed by the software. It also requests user input. The ”Data Management Layer”, which is normally located on server side, is responsible for storing data and replying to queries. The centralization of this layer on a server guarantees a consistence of the data. Finally the ”Application Processing Layer”, which can be located on either side, has the computive power of manipulating the data and presenting results. Those layers are presented in Figure 4.10

He distinguishes between ”Fat Client” and ”Thin Client” Architecture. The ”Fat Client” architecture has the advantage that a lot of the processing is performed by the client. This results in a less heavy processing load for the server and in a lot of cases in less network traffic. The disadvantage in our scenario is that for allowing the client to

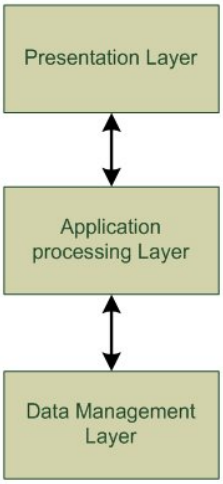

Figure 4.10: Application Layers. [Som04]

process the information, it is necessary to transfer all data to the client. As discussed before, this can contain too many pieces of information in case of large ontologies. The "Thin-Client" leaves all data management and application processing on the server side of the system, and transfers only necessary information and results. This results in a heavy processing load for the server, but reduces the network traffic sent through the wireless network. [Som04]

Using a thin client architecture would also allow creating client specific applications for plenty of different platforms much easier than using a fat client architecture, as all functions would not have to be implemented on all mobile devices' platforms again. The backend could be used by a Nokia phone running symbian OS, by Windows Mobile devices or even by a web server that offers a web version of the software simply by calling the server side functionality. [Som04]

This architectural approach can be described as "Two-Tier Client Server Architecture with thin Clients", as two sides are involved and the clients are "thin". Sommerville recommends this architecture for data- intensive applications with little application processing. As the limitation of traffic through the wireless network is one of the main goals of this application, it seems to be fitting into that recommendation. If we add the different service providers that actually provide the content to our view, we would implement a "Multi-Tier Client Server Architecture" that is recommended for applications where data from multiple sources has to be integrated. [Som04]

The interaction between client and server can be either synchronous or asynchronous. Synchronous interaction means that the client sends a request to the server and waits until it gets a reply to that request including data. The Server is blocked for that time

as well. Asynchronous interaction describes that the client continues his operations and does not wait for the server to return control. [Pre05]

As our application is reliant on the information transmitted by the server, it is necessary to receive a response as quick as possible. One of the main disadvantages of synchronous connections is that client and server are blocked until the request is terminated [Wue05]. In case of a connection loss, this request might take very long time until the connection gets reestablished. For the client this might be to neglect, as in case of a connection loss the client is locked anyway. For the server side this situation does also not apply, because every user has to use his own sets of objects on the server side, running in their own instance, as the objects are user specific. So the user will block the connection for his private session, not the connection for other users.

Synchronous communication can be realized using different models like , which is independent of the programming language, COM, which is based on Microsoft Systems, and Remote Message Invocation, which is based on JAVA.

CORBA is an Object Request Broker that acts as a middleware between distributed objects. Here objects can exchange information even if they are located on different Systems in a Network. They can also be written in different programming languages. This is realized by describing the objects using an interface definition language, an object request broker that manages the requests for different objects, translation services and a set of common components that may be required by many applications. [Pre05]

The main problem about shared objects in a mobile context is that there are some research projects only that implement that technology [FGM05] and also an implementation of Jini for Java [Jin09] but no approach is as cross platform as CORBA, which is not currently supported for all mobile devices. This practical limitation requires a different connection between clients and servers.

Another possibility for connecting clients and servers are web services, as web service support is widely implemented in mobile device operation systems [e.g. [Bus06], [Sad08]]. Web services allow as well the implementation on a website that allows access to the services using mobile browser. So they offer all connectivity required. How those services should be designed will be described in the following.

In this scenario there is only one web service provider - the server side of our system - that offers the required functions to the client side. This service can be identified using one URI that can be stored in the clients' application, so it is not necessary to use a service directory. The client can send all requests to the service and gets the results in an appropriate format.

As all information might be confidential, the call of the web service functions should generally be encrypted. It is as well necessary to implement a user management.

The encryption of a web service can be implemented in different layers of the pen System Interconnection Reference Model (ISO Layer Model). [Bur04] presents concepts for the Transport Layer and for the Application Layer:

Security concepts for the transport layer complement existing network protocols with additional security measures. and are widely used protocols on that layer. HTTP is established as a transport standard for web services and HTTPS is an extension to HTTP using SSL security standards. The SSL communication first requires an authentication of one or both parties. Afterwards data gets submitted using a secure channel. The cryptographic techniques supported by SSL guarantee confidentiality and integrity. One of the problems is that intermediaries, like forwarding web services, are able to read the information. As in our scenario there is a direct connection established which does not require intermediaries, this concern does not apply. [Bur04]

Concepts on the Application Layer are encrypting the message before submission. Burghardt recommends the encryption of an XML document using XML encryption for guaranteeing confidentiality. For integrity he recommends the use of an XML signature that computes an encrypted hash value of the documents that can be compared with the document received by the service provider. [Bur04]

As the whole topic of encryption would be a master's thesis of its own, I will use encryption using HTTPS for the application as it is implemented in most mobile devices and easy to use. The encryption of an XML File would require more implementation work, and as there is no intermediary involved, the easyer way of using HTTPS seems to be sufficient.

Another important topic considering client server communication is the user authentication and the session management. User authentication can be achieved by implementing the submission of a username and a password. This can be implemented in each function by adding a user name and password variable. As we already discussed to use HTTPS on the transport layer, a further encryption of those login credentials is not required. [Esp05]

In the ancient times, web services were implemented stateless. That means a service receives a request, performs a task and returns a result. No further information that could be used in the next session is stored. That scenario was acceptable for designing web pages. For more complex services it is desirable to add a state that can be included in requests. Nowadays web services can be implemented stateless, conversational or stateful. Stateless services allow message exchanges based on the content of the input

message. Conversational services can use prior messages for the computation of results. This messages are based on a logical sequence. A stateful service acts upon stateful resources that are stored on the service provider side. This is required for our scenario. Conversational and stateful services require a session management. [FFG+04]

In asp .net for example, the session management is implemented on two different ways. First there is the possibility to use cookies that get stored on the client side and return a unique id that redirects the user to his session, where session objects are kept alive for a defined amount of time. In some scenarios Cookies are not accepted by a client. Therefore it is also possible to include the Session ID in the URI of the service. In case of clients that request an XML web service this session ID can also be included in messages that are sent for requesting the service. The only requirement that has to be fulfilled is to add the session ID to the SOAP requests sent from client to server side. This way objects will be kept alive for the defined amount of time and it will be a lot quicker for the client to request additional information from the service. [Esp05]

4.3.3 Further look on Web Services and SOAP

The description about SOA and web services in chapter 2 has been in a general way, describing service discovery and the general concept. For the implementation of web services the author wants to describe the SOAP protocol in a more detailed way, as it is used in the application.

The SOAP Protocol is used for submitting parameters to remote procedure calls and submitting back the return values. The other use for SOAP is exchanging random XML files. [Kue03]

A SOAP Message has a defined structure. First it has an envelope, that defines the SOAP protocol. Second it has a header that can contain information for different hosts, which have to compute results in a special order and then forward the message to the next host. In case one service cannot provide the services for some reasons, it can place error codes in the header. [Bur04]

The body can contain any kind of XML code. In case of a remote procedure call the service provider has to define what values must be submitted and in which way they have to be formatted. SOAP does not have a definition on how to transport the messages. In theory it would be possible to send floppy disks via Mail. Normally remote procedure calls are executed using http or https. [Kue03]

For performing remote procedure calls in our application, the required variables have to be submitted using SOAP messages. Those variables will be extracted on the server

side and the results will be transferred back.

An example for a SOAP message for the Service GetValuesOfProperties is given in Figure 4.11:

```
<?xml version="1.0"?>
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
   <s:Header>
   </s:Header>
   <s:Body>
      <sessionID>AA9FE23423F3F553</sessionID>
      <requestedFunction>GetValuesOfProperty</requestedFunction>
      <Property>Doctor<Property>
      <startswith><startswith>
      <part>1</part>
   </s:Body>
</s:Envelope>
```

Figure 4.11: SOAP Message for GetValuesOfProperties().

Here the body contains the SessionID, the requested function, the selected service, the selected property and the part of the result list requested. This information is extracted by the server side and forwarded to the actual functionality, provided by objects on the server side. As there is only communication established between two parties, the header is not required and stays empty. The envelope is used for describing the used SOAP standard.

4.3.4 List of the required Web Services

As the previous subchapters described some required functions, this section will give a detailed list of all required services. All services are accessed using SOAP. The general functionality of SOAP has already been described in the previous subchapter. The login service and the session handling service are normally provided by the programming environment. In the last subchapter an example for Microsoft ASP .net was provided. Other programming languages offer similar services. I will not further describe the implementation of those services, as they are not very relevant to the goal of this thesis.

All requests by the client need to contain the Session ID, provided by the system after the login service has been completed. This will not be further described in the concrete services.

As recommended for designing web services, the services are named in a way they describe the functions from the client's point of view. GetUserInfo for example describes, that the client gets the user info. The service sends the information and does not re-

ceive it. The services are structured into Administrative Services, Ontology Services, Data Transfer Services and Services for adding Elements to Ontologies.

4.3.4.1 Administrative Services

GetUserInfo()

Requires a username. Returns the available user information (e.g. name, address, etc.) to the client.

SetUserInfo()

Requires updated user information in an XML File. Updates the information in the system.

GetAvailableServices()

Requires a username. Returns a list of available services.

GetSubscribedServices()

Requires a username. Returns a list of subscribed services.

AddSubscribedService()

Requires a new Service the user wants to subscribe, including user information for that service. Returns a confirmation if the service is stored.

RemoveSubscribedService()

Requires the service the user wants to cancel his subscription. Returns a confirmation if canceling was performed.

4.3.4.2 Ontology Services

GetThing()

Requires the selected Service, a letter combination of the starting letters of a Class or Object and the part of the result list requested. Returns a list of the found Classes and Objects.

GetClasses()

Requires the selected service, a letter combination of the starting letters of a class and the part of the result list requested. Returns a list of the found Classes.

GetProperties()

Requires the selected service, a letter combination of the starting letters of a property, the class the property is describing, and the part of the result list requested. Returns a list of the found properties.

GetValuesOfProperties()

Requires the selected service and a property, as well starting letters and part of result list. Returns a list of the found values.

GetObjects()

Requires the selected service, a letter combination of the starting letters of an objct and the part of the result list requested. It can also use properties and property value pairs. It returns a list of the found objects.

4.3.4.3 Data Transfer Services

SendData()

Requires the selected service. It further requires the uploaded data and the attached Metadata. It returns a confirmation if the data was received and forwarded to the selected service.

4.3.4.4 Services for adding Elements to Ontologies

GetRequiredInformation()

Requires the selected service and a class the new object shall belong to. It returns an OWL-Class definition with all required and optional fields for adding a new element.

SubmitNewElement(OWL)

Requires a OWL Object Definition and a Service. It returns a confirmation if the object was received and forwarded to the service.

4.3.5 General interaction Model

UML Sequence Diagrams identify the message exchange between objects and entities. They show the exchanged messages referring to the time they are exchanged. The Sequence Diagram gives an overview of all communication required in the system. Those diagrams can be used in the design process for modeling the interaction between the different components of the system. [Kec09]

Figure 4.12 illustrates this communication process, using the above described parts of the system and services. This diagram is supposed to give an overview about the different functions.

It describes the use case ”complex tagging”, where the user selects a concept, a property, a value and finally an object. All message exchanges for selecting tags are multiply performed, as the user might enter a few starting letters or correct his inputs. In case of the use case ”simple tagging”, the process would be limited to one message exchange GetThing().

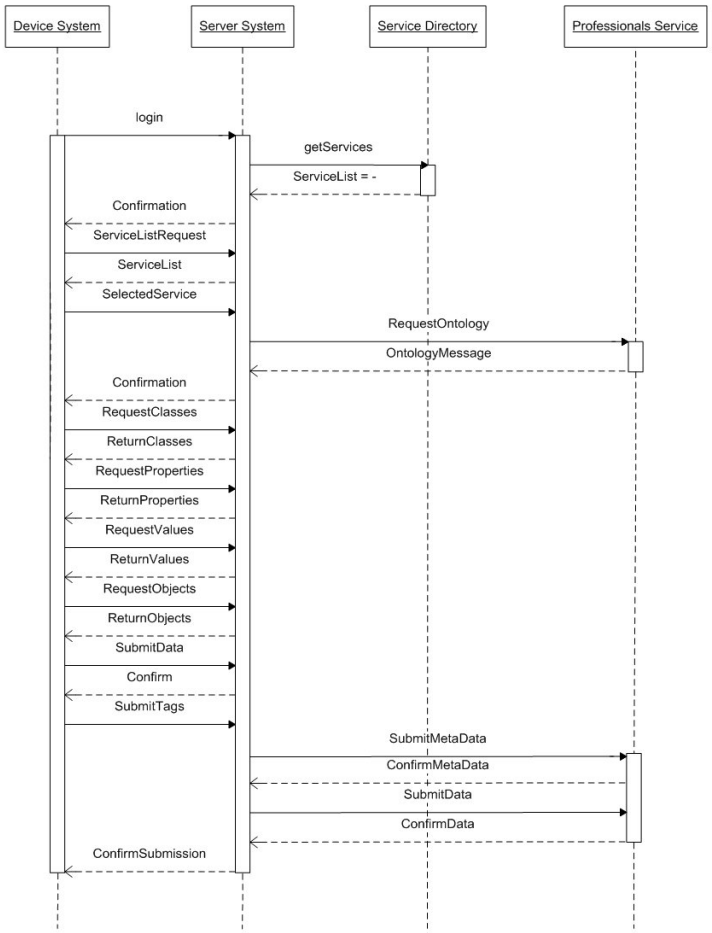

Figure 4.12: UML Sequence Diagram.

4.4 Object Models on the Server Side

As the main business logic of our application is provided by the server side, I will discuss an object model for this side of the application first.

Modern programming languages are object-oriented. Languages used on the internet are e.g. Microsoft ASP .net or Java. Therefore an object-oriented model is required. Object-oriented systems have the advantage of being easier to change as objects can be seen as standalone entities. They can be easy modified or exchanged. [Som04] This Chapter is not going to give a general introduction on object orientation, as this topic is widely discussed and well known.

The Model Driven Architecture Approach suggests two levels: an implementation independent level and an implementation dependant level, which is a more detailed, platform dependent level [Som04]. This chapter will just describe the implementation-

independent level. Some details about the implementation will be given later.

Sommerville suggests to start by identifying the principal objects in the system after having designed the system architecture. As a next step he recommends to develop design models and finally specify object interfaces. [Som04] I will follow this recommendation in the following sections.

4.4.1 Object Identification

I start by identifying the possible objects. A possible object scenario is presented in Figure 4.13.

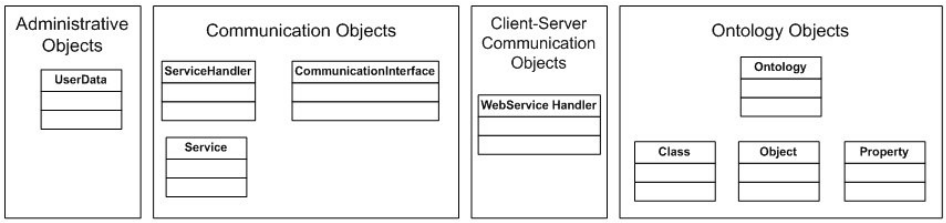

Figure 4.13: Object Identification.

For better illustration of the object discovery process, I divided the objects into four groups: Administrative Objects, Communication Objects, Client-Server-Communication Objects and Ontology Objects. Further information on the functionality of these objects will be provided in the next subchapter.

The **Administrative Objects** contain one object only. It is called **UserData** and contains the Information about a user (e.g. name, username, password, etc.).

The Second Object Group includes the **Communication Objects**.

The first object is a **Service**. It has information about the attached Ontology, the URI of the service, and a Service Description.

The second object of this group is the **Communication interface**. It contains information about the services' interfaces. It provides the functionality of submiting data, metadata and new concepts or things to add to an ontology.

The third object is called **Service Handler**. It is a supportive object that works as a container object that handles all service objects.

The Third Group of objects, the **Client- Server Communication Objects**, contains the **Web Service Handler Object** for maintaining communication to the client. This object is normally provided by the programming environment and contains descriptions for all web services. It maps the services to the actual objects and their functions. The creation of web services was described in a previous sub chapter. This

Communication Object is a representative for the functionality provided by the system, so it will not be further described.

The fourth group of objects contains the **Ontology Objects**. As main ontology object I identified the **Ontology**. It has to contain all information stored in an ontology and has to provide as well the functionality of allowing queries considering the content of the ontology. This Ontology Object contains **Class**-, **Property**- and **Object-Objects**, which are also objects in this last group.

4.4.2 Object Design

In next step I will design the objects and interfaces. Therefore I will first discuss about the Ontology Objects, as they seem to be the central point of the object model. Afterwards I will describe communication- and administrative objects.

4.4.2.1 Ontology Objects

Even if the object design is done before the actual implementation of software, it is useful to investigate existing systems, used for accessing ontologies through an API. I will do that in a first step for getting indications for developing our class structure. It also does not seem to be useful to develop a complete as there are existing APIs we could use for our Software. This is done in the first step of this part.

In the second step I will actually design the Ontology Objects.

Ontology APIs There are several APIs available for accessing ontologies. Some as Open Source software some as commercial software. In this subchapter I will present a small selection of APIs.

The Jena Ontology API [Jena] is a Programming Toolkit that allows the use of ontologies in JAVA. Jena is based on RDF and so it supports all RDF based languages like RDFS, OWL and DAML+OIL. Jena tries to realize a consistent programming interface not differentiating which ontology representation language is used. As the different languages vary a lot in their expressive power, Jena introduces different profiles that regulate the possible expressions. [Jenb]

As it is possible to have an object in an ontology being e.g. a class and a property, it is difficult to map this polymorphism to Java Objects that cannot be dynamically changed. This is realized by creating many Java objects for the same OWL object, depending on the use of the OWL object. [Jenb]

It is supported to import different ontologies and there is also reasoning support. It is possible to read ontologies, get all classes and create Java objects. After that, it is possible to read the instances of the classes - the OWL objects - and create Java objects as well. Now JENA can read the properties and Property-Objects are created in Java. [Jenb]

Once the Object Model is constructed in Java, it is possible to easily access Classes, Objects and Properties. It is supported to select certain objects, for example instances of a class or objects that have a certain property. Further is supported. SPARQL is a query language that is similar to but used for querying RDF documents. Those two ways allow an easy access to ontologies in Java. For summarizing: The Jena Object Model contains plenty of objects for reading the files, writing, allowing queries, etc. but the main objects are the ontology objects, classes and properties.

Querying the semantic web or ontologies allows a more complex structure than sending queries to a relational database. An example for such query would be "If Peter can choose between Coffee and Cappuccino, what will he choose?" with the additional information provided in an ontology, that Peter is allergic to Milk and that Cappuccino contains milk there would be just one possible answer.

OWL-QL is a powerful language for querying the semantic web and ontologies. A query using OWL-QL contains a Knowledge Base pattern, which can be one or more knowledge bases or variables. As answers to queries can be very complex they are split in answer bundles. The client can either ask for more answers after receiving an answer bundle or terminate the request [FHH04]. Figure 4.14 illustrates this query process.

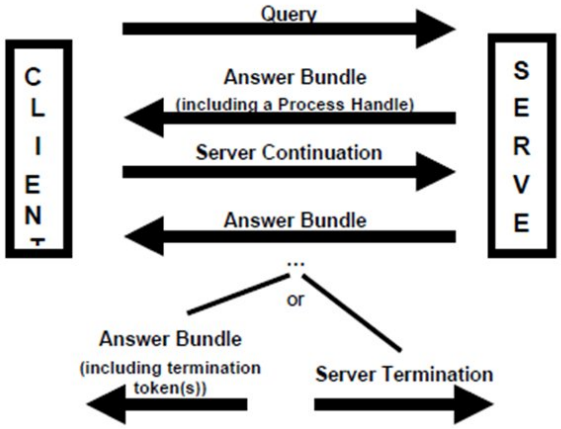

Figure 4.14: Client Server Communication in OWL-QL. [FHH04]

The OWL-QL is not yet completely implemented but partly used in a lot of projects [ZHA05].

Zhang [ZHA05] compares five query languages in his thesis considering their expressive power and their performance. The result is that RDQL is one of the easiest languages, as it has an SQL like syntax. SPARQL, which has not been fully implemented in 2004, has the same advantages and even a larger expressive power. RACER was the fastest of the tested query languages. [ZHA05]

This discussion of query languages presents two conclusions: First: The object model for our application should contain an ontology class that provides an interface for accessing the ontology. Second: There are already fully implemented ontology APIs that could be used in our application.

Ontology Object Model As discussed above, the ontology object model contains one class called "Ontology". This class provides a central access point for submitting any requests to the objects that are part of the ontology. Those objects can be Classes, Objects and Properties. Figure 4.14 shows those classes including their interfaces and variables:

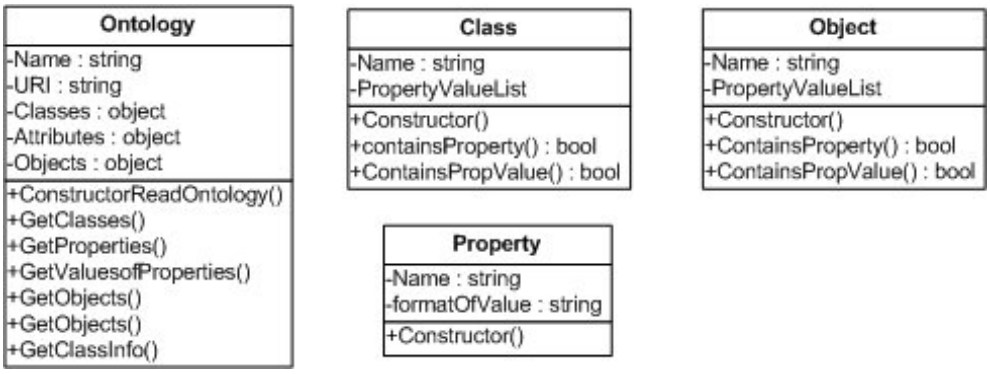

Figure 4.15: Ontology Classes.

The Ontology class, illustrated in Figure 4.15, contains the following **variables**: First there is a name and the URI of the Ontology, used for description and identification. Second there are Class-, Attribute- and Object- Objects that contain a representation of the available resources described in the ontology. The Ontology Class contains a Constructor, that reads the OWL or RDF files and creates all required objects.

In Figure 4.9 I introduced the different functions GetThing, GetClass, GetProperties, GetValuesofProperties and GetObjects. Those functions are now assigned to the Classes and described below:

getThing(string startsWith, integer Part): Array [string, string] Things, integer part, integer parts

This Method requires a parameter "**startsWith**" which contains the first letters of the thing searched. The parameter "**part**" gives the number of the result set the user wants to get displayed. Those parameters are the same for all following methods and will not be described in the further sections.

The Method returns an integer "part" that contains the part of the result set submitted and an integer "parts" which contains the number of result sets. Those variables are also used in all following functions and won't be described in the further functions.

The method also returns an array containing the names of all Classes, Properties and Objects of the ontology that start with the given string as well as their type (Class, Object, Property). This array can be used for searching anything in the ontology and might be the start of the search of a user.

GetProperties(string startsWith, integer Part, string Object): Array [string] Properties, integer part, integer parts

This function requires a string "Object", which contains the identifier of the Object or Class of which the properties shall be returned. The requested part of those properties is returned in the Array "Properties".

GetValuesOfProperty(string StartsWith, integer Part, string Property): Array [string] Values, integer part, integer parts

This function requires the relevant "property" the user wants to get the existing Values for and returns an Array of "Values" for that Property.

GetClasses(string startsWith, integer Part, array [string, string] PropValue): Array [string] Classes, integer part, integer parts

This function requires the above described parameters. It can as well handle property-value combinations. So it would be possible to search for classes having the property "is_a" and the value "thing". This would return any class, as all classes are based on OWL Thing. It could also search for classes that have the property "wheels" and the value "4".

It returns an array which contains the requested part of all Classes.

GetClasses(string startsWith, integer Part): Array [string] Classes, integer part, integer parts

If no property is required the same function can be called without a "PropValue" Parameter. This is realized using this overloaded function.

GetObjects(string StartsWith, integer Part, Array [String, String] PropValue): Array [string] Objects, integer part, integer parts

finally returns all Objects, containing the selected Property Value Pairs that are requested in the parameter PropValue. Like this it would be possible to search for all

red Mercedes cars using the Array [is_a, car / has_colour, red / has_brand, Mercedes].

GetObjects(string StartsWith, integer Part): Array [string] Objects, integer part, integer parts

is also overloaded and does not require any Properties in that version.

As described above all Get-Methods require the string-parameter "StartsWith" that submits the first letters of the requested results and the integer-parameter "Part" that gives the part of the result list that should be submitted. The first parameter allows browsing the ontology by typing the first letters of the requested result. The second parameter allows splitting the result list into parts for reducing the ammount of data transferred to the mobile device.

One last method required is used for submitting class information to a user that wants to create a new object for an ontology. This information has to contain the mandatory and facultative properties as well as their formats. This is realized by submitting the class information. Objects in ontologies can also contain properties not specified in a class, due to interoperability issues it is not desirable to allow the user to add freely chosen properties. This function requires a lot of logic, as Ontologies can contain many different objects with different restrictions. A very limited example could be implemented like:

getClassInfo(Class): Array [string: Property, string: Type, string: Restrictions]

The returned array has to list all defined properties. The first Value of the array contains the identifier of the Property. Two examples: If it is an object it hast to return the reference to the required property, the type "Object", and the restrictions "". If it is an Integer, it has to return the reference the property e.g. "children at home", the type "Integer" and the possible restrictions in an OWL Format, like (0<X<10). Similar restrictions are suitable for strings.

As the development of an Object Model for supporting this functionality for all possible properties might be the topic of a thesis of its own, this example shall be enough for illustrating the overall meaning and giving an idea of a possible implementation.

The next class in the model is the **Class-Class**. It contains the following variables: The Name is a unique identifier, that represents the class. The Array [String, String, String] "PropertyValueList" contains the name of the assigned properties, their Values and the existing Limitations for assigning a new Value.

It provides the following Methods:

The **Constructor**(String: Name, Array [String, String, String]: ProperyValueLimitations)

creates the class, with the given variables.

ContainsProperty(String: Property) returns true, if the requested Property is implemented.

ContainsPropValue(String: Property, String: Value) returns true, if the requested Property value pair is existing.

The **"Object"-Class** contains the same variables and methods as the "Class-Class".

The **Property Class** has the variables name (String) and limitations (String). Limitations are given in an OWL syntax and required for adding new objects to the ontology. It contains a Constructor(string: Name, string: Limitations) that creates the property.

The above described functions cover all required search options. Parts of their actual implementation will be discussed in Chapter 4.6.

4.4.2.2 Adminstrative Objects

The administrative objects in our object design just contain one object, the object UserInfo. This is illustrated in Figure 4.16.

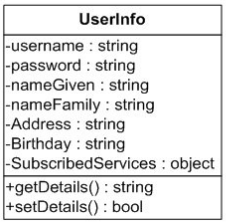

Figure 4.16: Administrative Classes.

The object **UserInfo** contains variables for a username, a password, a given name, a family name, an address and a birthday. There is also a list "SubscribedServices", which contains the services a user is subscribed to. The username and password are used for logging into the server side system. The username and password for a special service are stored in the service object, as them may vary for every service. The methods get and setDetails are used for updating user information and should be read as get/ set username(), password(), Birthday(), etc.

4.4.2.3 Communication Objects

There are three objects responsible for communication. The first one is a Service Handler, which acts as a container for all available services. The second one is a Service Object, which contains all information on the different services. The third one

is called SubmitInformation used for submitting information to a service provider and provides methods for this functionality. Those objects are visualized in Figure 4.17.

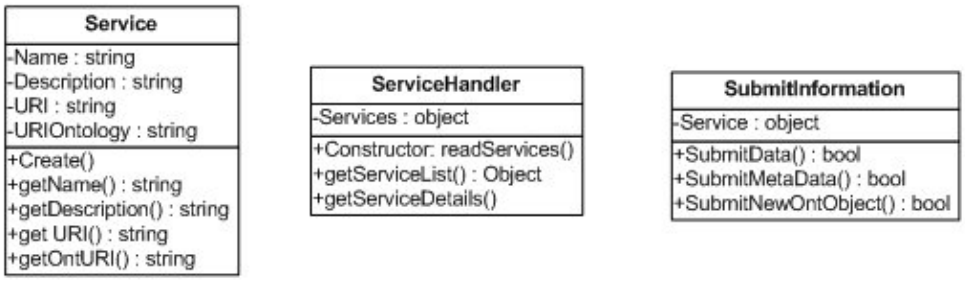

Figure 4.17: Communication Objects.

The **Service** Class contains the following variables: Name, for the name of the service, description, for showing the user what the service is about, URI for storing the services address and URIOntology for storing the URI of the ontology. The variables username and password contain the login information for the service. The constructor Create() requires values for all variables as parameters. The get operations return the variables for further use.

The **ServiceHandler** Class contains a list of service objects. It has a Constructor() that recives the service names and all details from the service directory and creates a service object for each service. The method getServiceList() returns a list with the names of all services. Using these names, it is possible to get further information addressing the selected using the getServiceDetails(servicename) metod.

The **SubmitInformation** class needs to be constructed using a service name. This service name allows getting all further information for submitting the tagged data. The method SubmitData(Data) submits the actual data uploaded by the mobile device. The method SubmitMetaData(Tags) submits an XML file containing all tags and user information.

The method SubmitNewOntObject(OntologyDetails) submits an OWL/ RDF File with extensions to the ontology created by the user. Therefore it uses the inputs the user has made. Those ontology details require a lot of logic, that validates all the input. This process will not be further discussed due to the limited scope of this thesis.

4.4.2.4 Object Model

Figure 4.18 illustrates the Object Model:

The Services are bundled in a Service Handler. The Submit Information Object

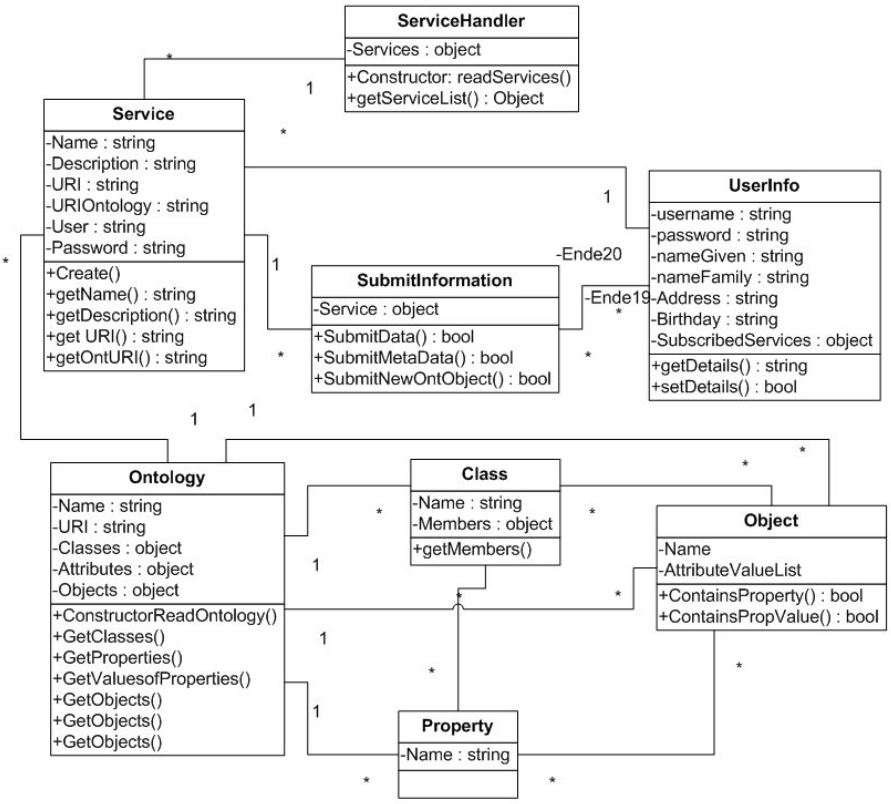

Figure 4.18: Object Model.

refers to one Service and to one User Information Object. Many Subscribed Services are referenced in the UserInformation Object.

One Service uses one Ontology that can be used for tagging, but the same Ontology can be used by many Services. The Ontology can be extended by other Ontologies by referring to them in the Ontology file. To the system this will be visible as one ontology only.

One Ontology contains many Classes, Objects and Properties.

One Object can be part of many Classes, and one Class can contain many Objects. One Class or one Object can have many Properties, and one Property can be part of many Classes or Objects.

As the Client-Server-Communication-Object can be seen as a helping object, that transfers requests from the client side to the server side and reverse, it is related to all Objects. It is not part of the logical view of this application so it is not represented in the object model. It forwards requests to different objects, by calling their functions. It is one abstraction level above the actual Object model.

The Server Side System can be described using a UML Component Diagram that

illustrates the connections to the surrounding system parts. This is illustrated in Figure 4.19:

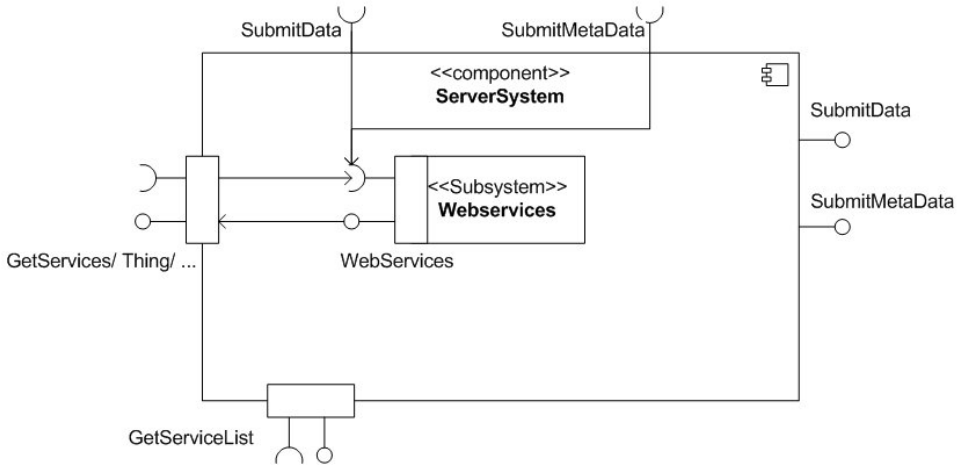

Figure 4.19: UML Component Diagram Server System.

The Server Side Component offers a socket to the client side for receiving data and metadata.

It offers a complex interfaces for receiving a service list from the service directory.

It also offers an interface for sending Data and metadata to the Service Provider.

It provides another complex interface that represents all GetFunction() calls (e.g. GetServices(), GetThing(), etc.) from the client side. Those functions are provided by a subsystem "Web Services" module.

4.5 Additional Functionality on the Client Side

Most of the operations are performed on the server side. The client side only needs some functionality for accessing the server side web services. The user interface has to provide required inputs and display the returned results. The client objects have to submit requests to the server, receive the answers and pass those answers to the GUI. Figure 6.14 illustrates this application design.

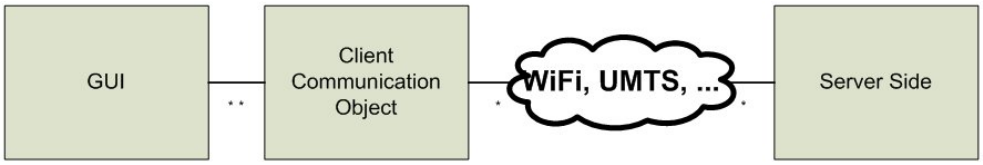

Figure 4.20: Application Layers.

This can be realized by using a communication object that allows the call of the web services. This object is illustrated in Figure 4.21 and will be described in this chapter.

Basically this object is a mirror to the server side functions, as it offers the same functionality that is also offered by the web services. It has to convert the information provided by the GUI to a SOAP message format and send it to the server. Further it has to receive the answer by the server and transfer the received XML file to a format the GUI can display.

Those mirrored functions are not further described, as they would look the same as already described twice in this thesis. Later I will give an example for one of those functions.

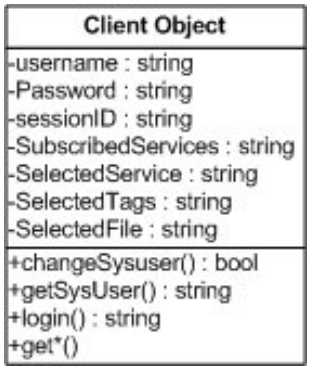

Figure 4.21: Client Connection Object.

The client object needs to contain the username and a password that is stored on the device. It gets read once the program is started and the object is created by the constructor.

It also contains a session id which is returned by the login method from the Server. It has a list of subscribed services that is returned from the server. The selected service is stored, once the user has chosen his selected Service. The selected Tags are a list of Tags selected by the user. The selected file is a reference to the file the user has selected for upload.

The Methods of that object are:

GetSysuser() returns username and password for accessing the System. SetSysuser() changes this information. This information is stored on the Device.

It further contains all Methods that are provided by the web service as a mirror. These Methods are listed in chapter 4.3.2. The difference to the web services is that the required information is transmitted using variables that get transformed to SOAP messages. The returned information gets extracted from the messages and forwarded to the GUI.

I will demonstrate this functionality using the method GetProperties():

The user requests the properties for a class. So the method gets called:

GetProperties(class, starts_with, part)

The class could be ”Thing”, startswith ”ha”, part ”2”.

The Method would use this information and transform it to a SOAP message, include the sessionID and send a request to the Server Side Web Service. The server would perform the required tasks and would return an xml file, using SOAP, containing the requested information. This XML file would contain a list of properties like ”handles”, ”has_child”, ”has_home”, etc. The system now has to generate an Array of String that uses the information sent by the server. This Array is forwarded to the GUI that presents it to the user.

The Clients' System Application is illustrated in a UML Component Diagram in Figure 4.22.

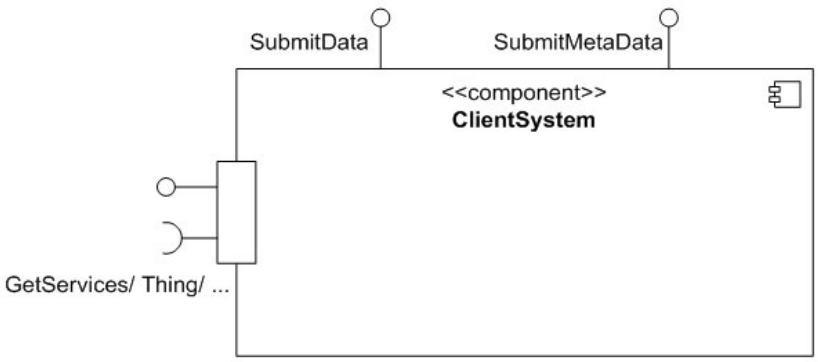

Figure 4.22: UML Client System Diagram.

4.6 Thoughts about the Implementation of some Server side Functions.

In this section I will discuss the implementation of some of the functions used in the ontology object. It will be done in a general way that gives the reader an idea about how these search operations can be performed. The goal of that section is not to produce executable source code, but to produce pseudo code that gives a more detailed explanation of the object model. As this thesis is not explicitly about ”modeling ontologies in object oriented programming languages”, this short description should be enough for understanding the general idea.

The first function I will describe is getobject():

Here the user gets a list of all objects starting with a certain letter combination. The ontology-Object searches through all Classes and Objects for matching "Things". The possible code is illustrated in Figure 4.23.

```
Integer Part = (part of function call) -1;
String searchfor = (part of function call);
Result = array [string];
Foreach object as element in self.objects              //foreach object in list, as element
     {
     If element.name startswith searchfor
          Result.add element.name;
     }
Result = Result ordered alphabetically;
Result = Cut (Part x 8) from beginning of Result      //removes beginning
Result = Cut ((Part + 2) from end of Result)          //removes end
Return Result;                                        // just the requested part.
```

Figure 4.23: Esample Code: GetThing().

Instead of using the foreach statement this function could also be implemented in SPARQL using the select statement illustrated in Figure 4.24.

```
PREFIX dc: <http://asdf.de/exampleontology>
SELECT ?name
WHERE  {
     ?x dc:name ?name
     FILTER regex(?name, startswith)
     }
```

Figure 4.24: Esample Code: GetThing(), SPARQL.

Searching for an Object with one property value pair would be performed with the foreach statement presented in Figure 4.25.

```
Integer Part = (part of function call) -1;
String searchfor = (part of function call);
Array [String, String] propertyValue =  (part of function call);
Result = array [string];
Foreach object as element in self.objects              //foreach object in list, as element
     {
     If element.name startswith searchfor
          {If element.ContainsPropValue(propertyValue)
          {Result.add element.name;
     }}}
Same manipulation of Result as before.
```

Figure 4.25: Esample Code: GetThing(), including Prop-Value Pair.

Here the function ContainsPropValue(PropertyValuePair), that was defined in the Object Class, gets used. If an Object contains a Property Value Pair, it returns true.

In SPARQL this function could be implemented using the WHERE clause adding the information that property x would have to require value y.

Due to the much easier implementation, the Ontology Class should be implemented using one of the above described Ontology APIs. Especially Jena2 combined with SPARQL offers a powerful search utility.

4.7 Other possible Implementation Scenarios

In this subchapter I will demonstrate some other possible implementation scenarios. In the implementation suggested in this chapter, i suggested the use of web services, requesting XML Remote Procedure Calls, using SOAP. There are some other technologies that might allow a different approach to that scenario. I will briefly introduce those technologies. Using XML for transporting the required information to the device is realized by sending a SOAP message from Server to client. Another way of submitting that data is using a "pull based" parser. One possible parser is kXML. It is running in a Java J2ME envi-ronment and makes it possible to pull XML information from a server. This might under some circumstances fasten the process of receiving required information. [DT]

One other approach, already mentioned above, is using shared objects, using an object request broker. I already introduced Jini for JAVA, which is available for the J2ME framework - a JAVA framework on a mobile device. One of the problems of shared objects is the complexity of setting up highly coupled systems. Especially in a mobile context this can easily result in problems, if connections are slow or not reliable. [onJ02]

Similar approaches exist also for Microsoft Windows Mobile Phones, using NET for Windows Mobile. [Mid]

Using AJAX in a mobile web application might also be a fast way of submitting the required information.

One last possibility would be to transfer the whole Ontology to the mobile device. In case of a rather small ontology, this might be possible very quickly. This would reduce the required interactions dramatically. In case of a large ontology and a slow internet connection this might take many minutes. A mixed scenario would be possible here - transferring small ontologies to the device, ontologies larger than a specific file size remain on the server.

After all the presented system architecture would not be very different using an Object Request Broker. The logic of the different functions would stay the same. The

only thing that would change is using a different connection from the mobile device to the business logic. I decided to use web services due to their interoperability. Further discussions on that subject will be done in the Outlook section of this thesis.

4.8 User Interface Design

As described in chapter 2, the general recommendation for designing user interfaces on mobile devices is allowing the user a wizard like input process that guides the user from step to step. The seven required screens will be described screen by screen. An example is given in the next chapter that discusses the actual implementation of the software.

The GUI of the application has to support a configuration view. This view has to enable the user to change the URI of the server side services. It as well has to enable the user to enter and change his username and password for logging in to the server side. As described above, this is the only information actually stored on the device. All other information is stored on the server side.

The user needs a screen where he can select his subscribed services. He has to be able to see them, add new services, modify them, and delete services he does not want to use anymore.

The above described functionality is more general and probably will not be used regularly. So it is possible to hide the configuration interfaces in a menu that offers the selections ”Configuration”, where the user can enter the above described information, ”Help”, where the user gets a manual for the software and ”Exit”, where the user can close the application.

The following screens are the main Part of the Application and oriented on the workflow illustrated in figure 3.7.

Screen 1: Home/ Select File:

This screen gives an input form, where the user can select the file to upload. He can browse the device for locating the file. He confirms and gets to the next screen by pressing the button ”next”. A possible illustration for this Screen is shown at Figure 5.4.

Screen 2: Select Service:

Here the user can browse his subscribed services that are presented in a list. The user selects the service he wants to submit his data. He confirms and gets to the next screen by pressing the button ”next”. A possible illustration for this screen is shown at Figure 5.3.

Screen 3: Choose Tagging Mode:

Here the user can choose simple or complex tagging. The differences are described in the next two screens. Therefore the user sees a screen with two buttons where he can choose his tagging mode.

Screen 4a: Select Tags Simple:

The user can start typing a tag, the system displays the available tags, starting with this letter or letters in a list. Once the user has located the desired tag can select it and it gets added to a presented list "selected tags". Each of the tags in that list has a button "delete" associated to it for allowing the user to remove a tag from the list. The user can add more information by selecting other tags. It should be also always possible to change to the "complex tagging screen" 4b. Therefore a button should be added to that Screen. Pressing a button "next" that will present the next screen 5.

A possible illustration for this screen, without the implementation of the "Change to Complex tagging mode" Button is shown at Figure 5.5.

Screen 4b: Select Tags Complex:

The complex selection of tags allows the user to browse the ontology by searching for classes and objects, by reference to their properties. Therefore the user first hast to select the class the searched object is part of. If he wants to search for any object he can select the class "thing", which is the top class of any OWL object. He can do that using a search field where he enters the first letters of the searched class.

In the next field he can enter the property he wants to search for. This also works the same way: He enters the first letters and gets a result set, where the possible properties are presented.

In the next field he can select the value of the properties the same way. In the last field he can select the actual tag from a result list, presented in the same manner as before.

Once the user has located the desired tag can select it and it gets added to a presented list "selected tags". Each of the tags in that list has a button "delete" associated to it for allowing the user to remove a tag from the list. The user can add more information by selecting other tags or press a button "next" that will present the next screen.

A possible input window for that search is presented in figure 4.26. The different steps represent the changes in the input screen. The Example shows the search for a doctor located in Espoo. The list beneath the input field sends the suggestions of the result set found in the ontology. The Input field allows entering the first letters of the selected thing. The List of chosen Tags on top shows the tags the user has chosen before with the ability of deleting them by pressing the delete button symbolized by

the ”X”.

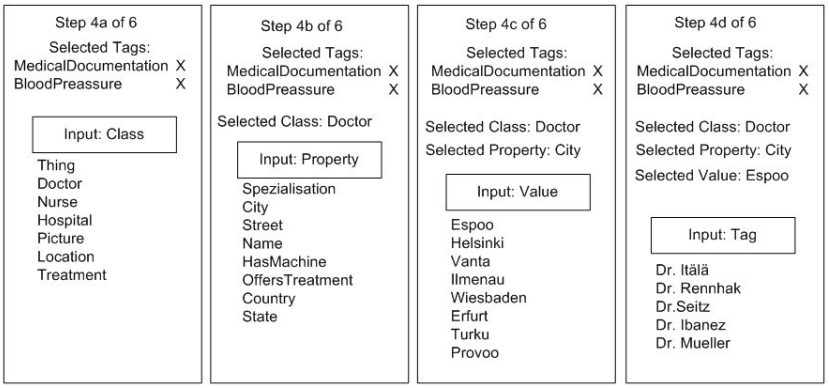

Figure 4.26: Advanced Tag Search.

Screen 5: Confirm:

The user sees the entered data, which contains the selected file, the provider and the metadata. Each part of the information has a button ”change” next to it that allows the user to get back to the screens 1, 2 or 3 for changing the information. If there are no changes required or all changes done the user can click a button ”Send” that starts the process of submitting the data and activates screen 6. A possible illustration for this screen is shown at Figure 7.6.

Screen 6: Sent:

If the data gets submitted properly, the user sees a screen that confirms the submission. In case of an error the user gets prompted an error message and sees a button ”try again” that returns him to screen 5 and a button ”home” that activates the first screen again.

In general I want to say, that the expert tagging is far more complex for the user. Most users might use the standard tagging, as it is much faster to add a tag. For some specific uses, where a user is not sure which tag to use, the advanced tagging mechanism seems to be suitable.

There also has to be a window implemented that allows the user to add a new object to a class. Therefore first the user has to be able to select the class the object is part of. Then there have to be input fields for all mandatory and optional properties. After selecting the properties the user can click a submit button and the new object gets submitted. It is required that all mandatory properties are filled in. A possible Screen for that purpose is presented in figure 4.27

As described in chapter 2, this GUI can be implemented on a local device running natively or using a virtual programming language like Java, requiring a virtual machine.

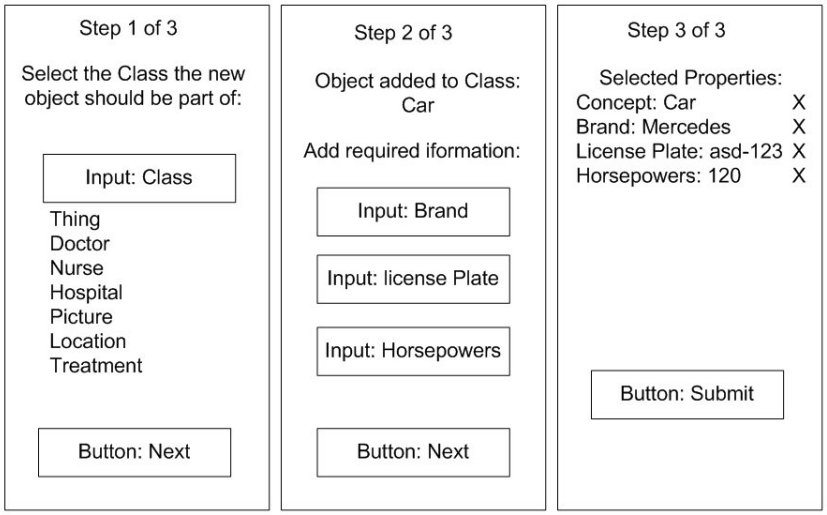

Figure 4.27: Adding Objects to Ontologies.

It can also be implemented as a web application, using a web browser running on the local device. The pros and contras will be discussed in the following chapter.

5 Presentation of results of implementation

5.1 Situation at the Start

As described in chapter 1 and Appendix B, one of the goals of the MyWellbeing Project is to enable the citizen to communicate with professionals. This communication is mainly concerning healthcare related information.

There have been implementations of two other software systems in the MyWellbeing Project before. They had the goal of allowing a patient to manage his medical information. The first software was LifeManager Karvalakki (finish for a "farmers head") that allows a user to manage his medical documents on a local PC. He is able to tag these documents, using a given vocabulary that has been divided into different categories. Afterwards he is able to browse his information by searching for tags. This software has been tested by a small group of patients that got their life- time medical information digitalized and tagged by researchers.

The second version of Lifemanager was able to manage this information online, not on a local PC only. It implemented a tag based concept for sharing information with family members and professionals. Professionals could access this information by logging into the platform and browsing the information shared with them. They can view files ordered by users or by the assigned tags. The access of the website is limited to a PC, as the website design is very complex. The tagging was realized using a pre defined vocabulary, having several groups of tags, similar to the vocabulary concept of LifeManager Karvalakki. This vocabulary is not based on ontologies.

The goal of the implementation of the current version - "Lifemanager Mobile Upload" - was to allow a user to send information to a professional using a mobile device, using vocabulary that can be processed by the receiving system.

This requires a shared conceptualization of the vocabulary used for tagging, which can be reached using ontologies. The used ontology can be provided by the professional that is supposed to receive the tagged document, and can be transmitted using one of

the earlier described standards for ontologies. This way interoperability with plenty of different systems can be achieved.

The implementation of the application is not supposed to be used in a real life context, so concerns like implementing secure, encrypted transactions are not relevant. It is supposed to be a demonstrative prototype that shows how citizens can communicate with professionals using a mobile device for delivering content to a professionals system. The professionals system was also not necessary to implement, as it seemed to be too complex installing and integrating e.g. a Hospital Information System. For the demonstrative purpose of this application it seemed to be sufficient to display the uploaded file and the attached metadata on a simple website. This way the actual delivery to a professionals system is simulated.

5.2 Web Application vs. local Application

The first decision during the implementation process was wether to design a web application, running on a server and interacting with the user through a web browser, or developing an application running natively on a small device. Both architectures were described in the last chapter and will be briefly summarized:

There are some advantages of an application running on a small device: Users are able to work using their device's "look and feel", implementing an interface that is similar to the programs they are already using. It also is faster to view software on a device than in a browser, as the software does not have to render the whole interface on every page load. GUIs of most devices also offer more functionality than the GUI supported by a mobile browser. It is also faster to start an application on a device than starting a browser, selecting the bookmarks and finally logging in to a website, providing user credentials.

The disadvantages of native software are that there are a lot of different operating systems on the market, all requiring different application formats and standards. The interoperability between those devices is very low, so for a cross platform - real world application it would require to develop many applications for different platforms. Also for the scientific goal of the implementation it would limit the actual test and presentation of the software to a small group of devices. The limitations of Java were discussed before.

The main advantage of a web application on the other hand is the high interoperability. Mobile browsers, offering many features (e.g. AJAX for generating content on the user side and reacting to user inputs quickly, java script for running parts of

the computation at the clients' side and flash for allowing graphically sophisticated GUIs) are available for most smart phone platforms. One browser that offers all this functionality is Opera Mini, which is available for Windows Mobile and Symbian OS [Ope09]. The Android browser offers similar features.

The interoperability was the main reason for implementing a mobile application running on a web server. One further reason was that the data exchange between client and server was much easier to implement using this approach, as the client side of the system, described in the last chapter, could be implemented on the same server as the server side. This way the whole communication process between client and server described there was not necessary to implement, as the server could directly use the objects, without using web services for communication with the client.

5.3 Development Platform

For realizing the implementation I chose Microsoft asp .net. It would have also been possible to develop this application in Java or PHP, as they also allow creating dynamic websites. Microsoft .net allows the use of Microsoft Visual Studio which is a powerful framework for software development. As I already had some experience in .net development, it seemed to be adequate to develop the application using Microsoft .net. Java has the advantage of platform independence, .net requires a Microsoft Windows Server. Due to a lack of experience programming in Java I chose .net anyway. A Microsoft Windows 2003 Server was provided so it was possible to run the implementation for tests on mobile devices.

5.4 Adaption of ONKI Server

During the development process we met the Semantic Computing Research Group of TKK [1]. They presented their "Finish Ontology Library Service" ONKI that is able to publish and implement thesauri and light weight ontologies for the web. One of the main goals of ONKI is supporting tagging with a standardized vocabulary, which is also a part of the topic of this thesis and which was also one of the main goals of this application. ONKI uses persistent URIs for identifying tags that means a tag is identifiable by the URI of its source file and the used concept. [TFVH09] So it is always possible to connect the tag to its source, as long the source is available online.

[1] http://www.seco.tkk.fi

For tagging support ONKI provides a lightweight web widget, generating general thesauri access into HTML. This widget allows searching for a tag, selecting it, searching for another tag, and so on. When the user has selected the desired tags it is possible to transfer those selected tags to other parts of an application, using HTML forms. ONKI selector does not provide support for searching tags by classes or properties. It is only possible to select tags by their name. Onki Browser offers a functionality that provides support for these requirements, but it is graphically too complex for displaying it on a mobile device. [TFVH09]

For allowing an easy implementation, that gives a demonstration on how tagging content from a mobile device could be realized, we decided that the use of ONKI Selector, which provides a web form with AJAX support, is sufficient. ONKI Selector can be easily implemented using an URI of SeCo Group, which includes some Java Scripts into the web application. One further reason for the implementation of ONKI was, that most users probably would not follow the very complex way of browsing ontologies by selecting classes, properties and their values for finding an adequate tag, as this process can be very time consuming. In a real life implementation it might be useful for some applications, but as ONKI Selector was already implemented and allowed the basic tagging functionality, it appeared sufficient.

5.5 Object Model

The object model of the implemented application was far less complex than described in the last chapter, as the whole ontology objects were not required as they were outsourced to ONKI Selector. In the following there will be a description of the used objects.

The main object of the Model is the XMLObject. It is used for creating the SOAP message sent to the service provider and also provides the functionality for reading SOAP messages. It is illustrated in figure 5.1.

This object contains variables for all required information that gets exchanged between client and service provider.

- The dateTime "`submitted_date`" contains the date- time value the data was submitted to the service Provider.
- The string "`used_vocabulary`" contains the URI to the ontology used by ONKI Selector.

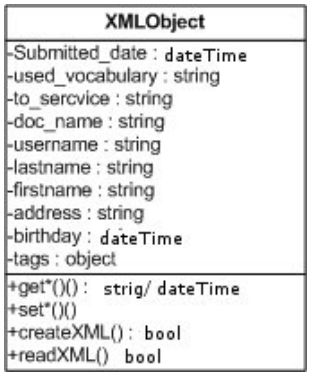

Figure 5.1: XML Object.

- The string "`to_service`" contains the ID of the service the information got submitted to.
- The string "`doc_name`" contains the document name of the submitted file; doctype contains its file type.
- The strings "`username`", "`lastname`", "`firstname`" and "`address`" as well as the dateTime "`birthday`" contain the self explaining information.
- The List<Sting> "`tags`" contains the tags added by the user.

All variables have a get() and a set() method for setting or requesting the information.

The function createXML(string folder, string filename) creates a temporal XML file that gets submitted later. Therefore it writes all information that is stored in the object to an XML file, that gets stored in the before specified folder on the server. The function readXML() is used for reading a SOAP message and is not used on the server. It gets used by the service provider application for reading all XML files that got submitted by different users. It is included in the object anyway for allowing a reuse of the object.

The second object is the "service" object. It contains all relevant information for a service as its URI, the used ontology, its name, and description. All information can be set and returned using set() and get() functions.

The "servicehandler" object contains all "service" objects. It provides functions for returning a list of all service names getServiceNames(), and it provides access to all information stored in the services using get() methods (e.g. getURI(serviceID)).

The constructor create(string ServiceXMLFile) reads an XML File, containing all

services and creates the service objects in that list. This allows the comfortable handling of all services.

The submission of the information is handled by the object ”SubmitInformation”. It has one function submitInformation(string ServiceURI, string XMLFileURI, string FileURI) that submits the previously generated XML file and the file to be uploaded to the service, specified in the function Call.

5.6 GUI Design

During the process of the GUI Design it was necessary to implement all functionality as lightweight as possible. Pictures were tried to avoid, and the used graphical elements are in a small file size.

When the user starts the website he first sees a welcome screen that tells him that he is now using the platform. This screen is shown in figure 5.2.

Figure 5.2: Welcome Screen.

In the next step, the user can select a service. For that he can use a simple drop down menu that offers him a list with all service providers. He can select one by clicking on it and he confirms his selection by pressing the ”Next” button. This screen is illustrated in figure 5.3.

In step 3 the user can select the file he wants to upload. The file is located using the upload file dialogue, implemented in many browsers. After selecting the file, he presses the ”Next” button. This is illustrated in figure 5.4.

Afterwards he can select the tags he wants to assign to his document. This tag list is generated by ONKI Selector, using the vocabulary provided by a service-specific ontology. He can select as many tags as he wants by typing the first letters of the tags in the Selector search field and clicking on the desired tag. Once the user has assigned all required tags, he presses the ”Next” button. This is illustrated in figure 5.5.

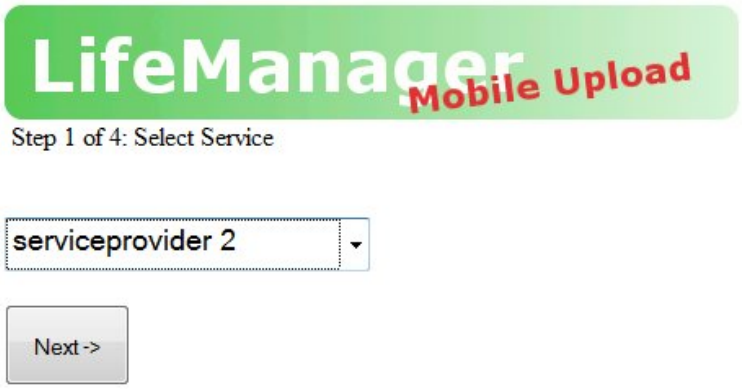

Figure 5.3: Selecting a service provider.

Figure 5.4: Selecting a file.

The next screen shows the selected service provider, the selected file and the selected tags. The user can edit the file, the service provider and the tags and proceed to send the information to the selected service as soon as no more changes are required. This change screen is presented in figure 5.6.

Once he confirms the input information, the information gets submitted and he can see a confirmation. If the information could not get submitted, an error message gets displayed.

The whole design implements one picture only, the buttons are large, so the user is able to select all information in a finger friendly way that does not require a keyboard. The user always can see at what point in the process of submitting the information he currently stands. This way he is always aware of how many steps are still required until the end of the process.

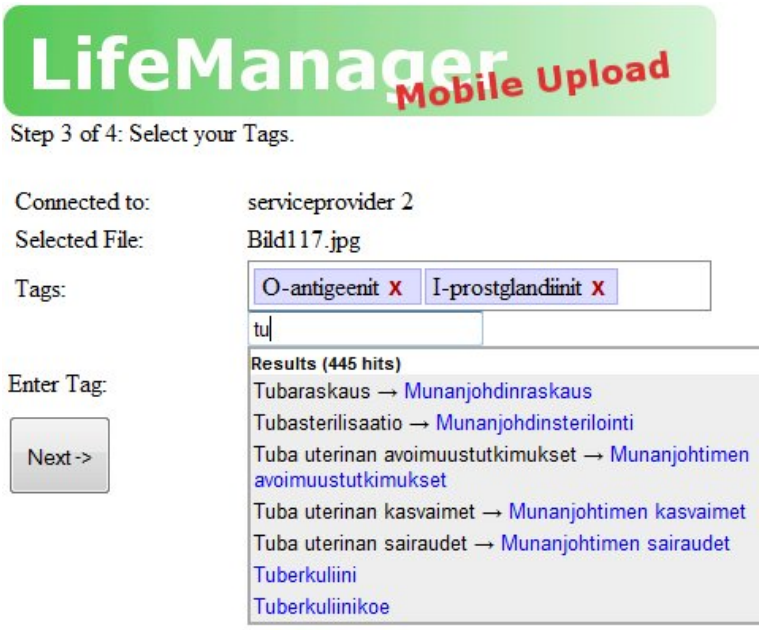

Figure 5.5: Selecting tags.

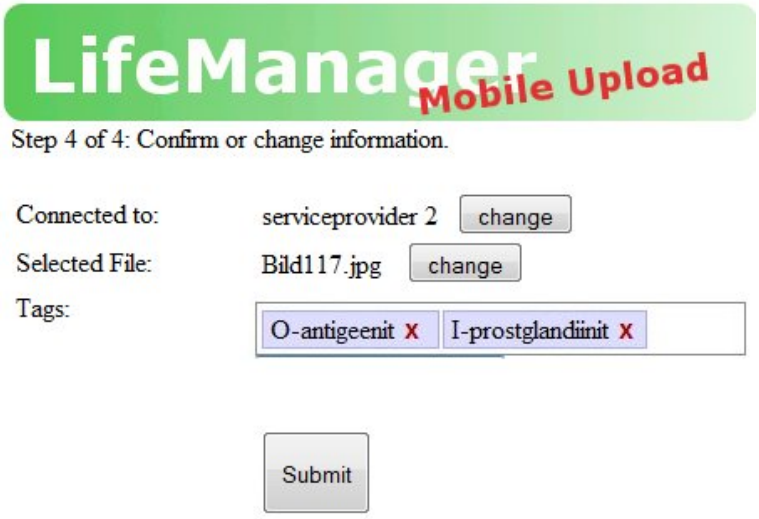

Figure 5.6: Edit or confirm information.

5.7 Web Service Design and Data exchange

There are several points in the application, where connections to different services are necessary. First when the user sees a list of his subscribed services, the real life application would connect to a service directory. The implementation of a service directory seemed to be out of scope for the use of this thesis. I decided to create an XML file that contains all information that would be transmitted by a service directory.

This information contains the name of the service, its URI, the URI of the ontology and a service description in clear text that could be shown to the user before selecting a service. The URI of the ontology is provided by submitting the required parameters for the ONKI applet. This file was generated manually and stored on the server the application is running. The implementation of a service directory would be possible by adding a service that provides the required XML file any time the user logs in to the system for receiving an update of new services. For the demonstrative purpose of this application, storing the file locally seems to be sufficient.

The next connection to services is required for transmitting the selected file to upload, and an XML file containing the user information as well as the required tags. This connection is realized by implementing a web service that simulates a professionals system. Therefore the system connects to this service and submits the file the user wants to upload as well as an XML file that contains all information provided by the user, including the server side stored user information (name, address, etc.) and the tags. This transmission is realized using a File Stream Reader, that establishes a connection to the web service and transfers the files. For guaranteeing that double file names do not create any conflicts, the service provides unique file names for all submitted information. In a real life application the professionals system would rename the files, store them in a professionally designed application and would extract all metadata and store it in databases or use it for further processing of the submitted data. For the purpose of submitting this metadata I used the XML data model provided in figure 4.8 similar to a SOAP message. The System creates this XML file and the service is able to read the file for extracting all information.

For demonstration purpose I designed a small server side application that reads all submitted XML files, presents the user information and the assigned tags and gives a link to the data submitted by the user. This way users can view all submitted information including meta data.

The mobile application is available at ”http://www.sommernetz.de/diplom/client” the server side application at ”http://www.sommernetz.de/diplom/service”.

6 Use Case 2: Ontologies for Communication between small devices.

6.1 Introduction

The second part of the research question deals with the question how ontologies can be used for describing the abilities of small devices and how those devices can exchange information on their abilities for communication purpose using ontologies.

Figure 6.1 illustrates a possible scenario for two mobile devices that connect and want to exchange information. The scenario could take place at a patient's home and the smart phone could be owned by a nurse, the PDA by the patient. The patient could have collected some information on his device, e.g. blood pressure values, received by his hemodynamometer. He now wants to submit this information to his nurse's device for further evaluation of the data, when she visits him. Therefore first a communications channel has to be established. Possible local channels are Bluetooth or IRDA (Channel 1), or a connection using a common Server via the internet using WiFi, 3G, etc. (Channel 2). Another possible device, not mentioned in the figure, could be a TV Set, providing the additional functionality of presenting information considering the patients health after receiving health data from his PDA (local or internet connection possible). This information could be extended by a web service that shows relevant information considering the health of the patient. For this example we will just discuss the two small devices.

Once the physical connection has been established, it can be useful to exchange information on the capabilities of both devices, e.g. the screen size, the possibility connecting to the internet, the file formats that can be handled or the allowed size of files to be submitted.

After exchanging this information the devices can establish their connection for interaction, based on the devices' capabilities.

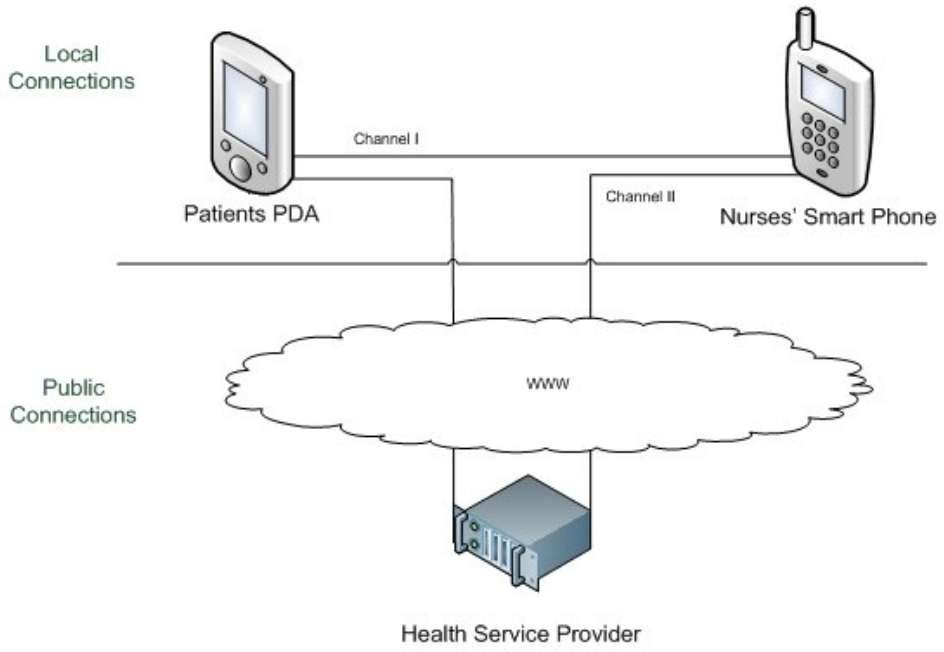

Figure 6.1: Scenario Use Case II.

As extensively discussed, ontologies allow the classification of things. It would be possible to develop an ontology that classifies the capabilities of mobile devices, using classes like "`data_connection`", "`output_device`" or "`video_support`", objects like "`screen`", which "`is_a`" output device, and properties like "`h_resolutuion = 360`" and "`v_resolution = 230`". It would be also possible to develop a class WiFi standard, containing multiple objects with different standards, so the device could refer to one of the objects for description of its own WiFi connection module. This would allow describing the capabilities of a device using a standardized format, which would enable an easy exchange of different modules.

This scenario would require that mobile devices are able to process the received information. They would have to provide at least some basic reasoning support. E.g. before submitting a file, the application would have to check, wether the selected file format is supported by the other device.

For examining this use case, I will first describe existing models for the exchange of device capabilities. Afterwards I will briefly discuss the two different connection channels, describing a Bluetooth connection and the connection via a central server. Here I will discuss the different communication layers the exchange of information between the devices could take place. After that I will discuss the requirements for reasoning support on the mobile devices. Finally I will give suggestions for an application

architecture.

6.2 Existing technologies for exchanging device capabilities

In this subchapter I will discuss scenarios, where mobile devices exchange information about their capabilities with a partner. In most existing scenarios the other side is a web server or another computer that tries to communicate with a mobile device. I was not able to discover any examples for exchanging details between mobile devices in literature.

In case of a web application, as we developed in the previous chapters, it could be useful for the server side to receive information of the devices' abilities for providing a device optimized version of the application. Considering the graphic abilities of a device, this approach is similar to WALL, a library to multiserve applications on the wireless web, described in chapter 4. As explained there, WALL detects the device and presents a version of a WAP page that suits the devices needs. WALL uses WURFL, a database that stores most devices abilities. Examples for those abilities are information on the CHTML engine of the device, the display, pdf support, streaming capabilities and Ajax support. There are APIs for three big programming environments for WURFL, Java, PHP and .net. [Pas07]

For detecting a device, WURFL uses the user agent string. A user agent string is an identifier that gets delivered any time a device makes a request to a server. This string is a unique id that allows identifying the exact model or version of the device making the request. [But02]

After detecting the user interface, WURFL is able to provide information on the device that allows the display of different versions of a website, depending on the information about the devices' abilities stored in WURFL. [Pas07] A possibility would be to display flash lite only to devices that support flash technology. The other devices could be redirected to a different version of the website. WURFL works on web servers only, and provides APIs for the above described programming languages. For solving the question how devices could exchange their capabilities, I will later discuss the use of WURFL for Bluetooth connections, requiring a version of the XML file, containing all information on mobile devices, on any device that wants to use WURFL. Afterwards I will discuss the different information stored in WURFL for specifying requirements on the ontology and the containing properties for exchanging abilities of mobile devices.

Martin et al. [MRM+08] describe several possibilities for identifying devices. Their use case deals with interaction of small devices in a vehicular network (a network established in a car). Those possibilities are:

Bluetooth device identification profile [BLU07] provides information on the vendor, model and Bluetooth interface of a small device in a "Service Description Profile". This profile is submitted while negotiating a communication channel between two Bluetooth devices, and gives enough information for collecting information on the devices ability from a database. [MRM+08]

Martin et al. [MRM+08] also suggest the use of synchronization Protocols like SyncML or Active Sync. Sync ML provides a Standard, defined for synchronization of mobile devices. Part of the Sync ML synchronization process is the device identification, which is illustrated in Figure 6.2. The information presented here would also be sufficient for requesting the devices capabilities from a Database.

```
<DevInf xmlns="syncml:devinf">
    <VerDTD>1.1</VerDTD>
    <Man>Nokia Corporation</Man>
    <Mod>Nokia 6820</Mod>
    <FwV>V 3.70</FwV>
    <SwV>V 3.70</SwV>
    <HwV>1701</HwV>
    <DevID>IMEI:XXXXXXXXXXXXXXX</DevID>
<!-- .... -->
</DevInf>
```

Figure 6.2: Partial Device identification using SyncML. [MRM+08]

Martin also mentions the use of WURFL, by forcing the mobile device to send an http request to a local server, run in the cars' system. [MRM+08] This approach does not seem to be useful for the goal of our use case, as it would require the implementation of a small web server on both mobile devices. In Martins use case, using a cars communication system, there might be more resources available.

Martin's last approach is sending a small software to the device, that extracts as much information as needed for the device application. This would require different software for all available platforms and does not seem to be useful in our context. The approach for detection Martin finally uses is trying all different strategies, until one provides identification. If there is no identification possible, the system creates an error message. In case of a positive identification, the system queries the devices identification number to a database and extracts all required information. Martins' algorithm was able to detect 92% of the tested devices. [MRM+08] Martins approach

is similar to WURFL, as it also tries to identify a device and extract information from a database afterwards.

Using ontologies for the description of devices would allow the use of a data base free environment for device detection. Here devices could submit an RDF or OWL File, including their abilities, to the other device. This way they could exchange information on those abilities.

CC/PP is another model used for describing clients' abilities for servers. It is similar to WURFL, but is built on RDF [KRW+04]. It provides a more general framework for the inclusion of new vocabulary for the description of other system parts, using RDF and RDFS statements [IRRH03]. A possible statement made in CC/PP would be that the screen component has a resolution of 440 x 300 pixels (ontology typical (subject, predicate, object) statement). The ontological design of CC/PP will be discussed later in this chapter.

Bandara et al. [BPRC04] suggest an ontology for the description of services and devices. In the case of a printer for example the location might be relevant for selecting a printing service, as the owner of the requesting device probably wants to collect his prints close to his location, not on the other side of the world. They suggest an Ontology containing five Classes: Device Description, Hardware Description, Software Description, Device Status and Service. Those classes are illustrated in figure 6.3.

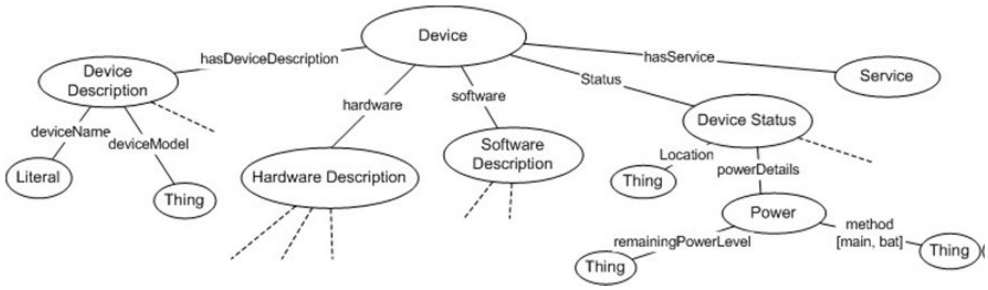

Figure 6.3: Device Ontology. [BPRC04]

These classes contain information about supported file types, locations, hardware properties, and many more details. As they are represented in OWL, it is easily possible to extend them. The difference to WURFL is that Bandara et al. have a wider focus, which is meant to describe any device with any associated service. Including services to the description is also a difference to CC/PP. The whole ontological design is trying to describe all relevant information in a device centric ontology. Therefore they provide an ontological framework that can be extended to any needs for describing devices. [BPRC04]

6.3 Communication Channels

In this subchapter I am going to discuss the different channels for communication. First I will describe a local connection (Channel I in Figure 6.1) using the example of a Bluetooth connection. Afterwards I will briefly explain how this scenario changes using a server as intermediary (Channel II in Figure 6.1). Therefore I will first briefly describe the Bluetooth protocol, then I will discuss the different layers of communication, referring to the ISO Model, on which the information exchange could take place.

The Bluetooth Standard was first published in 1999 and contained two documents: The Bluetooth Core Specification, which describes communication, protocols and technology for communication. The second document is the Bluetooth Profile Documentation which presents possible Profiles for different use cases. Bluetooth had the aim of providing a radio interface that should be small, low energy consuming and cheap in production and should act as a replacement for wire connections in a local area with a low range. Those networks are called Wireless Personal Area Networks. The different Bluetooth profiles support e.g. Intercom, LAN, Headsets or File Transfer. Currently there are three different versions of Bluetooth, Bluetooth 2.0 and 3.0 offer faster connection speeds. [GK09]

The Bluetooth Protocol is divided into different layers. Those layers will just be mentioned, as a detailed description does not seem useful for the goal of this thesis: Radio Frequency Layer, Baseband, Link Manager, Service Discovery Protocol and Logical Link Protocol. Those Layers are used for establishing a connection and exchanging packets. For actual communication, using the application protocol, Bluetooth implements adapted protocols of PPP, UDP, TCP/IP and WAP, as well as some special protocols for exchanging objects and files. Therefore Bluetooth implements the protocol OBEX as a standard. [GK09]

There are APIs for the main programming languages available, that allow the use of Bluetooth. The Java APIs e.g. allows: To register services, to discover devices and services, establish different connections between devices, using those connections, send and receive data, manage and control the communication connections and provide security for these activities. Using the API, the developer is able to build new Bluetooth Profiles that allow the use of Bluetooth for different scenarios. E.g. for File Exchange it is also possible to implement existing Profiles. [Mah03]

As Bluetooth profiles are rules for the behavior of a device, it would be possible to modify one of the existing profiles for file exchange for integrating the support of an ontological description of the abilities of a small device. When paring two devices, there

could be a profile, that allows the exchange of files, but before this is done, the devices exchange OWL or RDF Files including their abilities. Afterwards they could limit the allowed communication to the supported media types. Implementing this support on a profile level seems to be very complex. For further exchange of abilities it would also be possible, to exchange an OWL file on the application level. This would require two applications, establishing a connection and exchanging the abilities. Afterwards the applications could limit the further communication to the abilities of the other side's device. This would require that both devices actually run the same application. Especially for implementing advanced reasoning support, described in chapter 6.3, it seems useful implementing this procedure on the application layer. Further discussions on that issue will be held in chapter 6.4.

The second channel for communication is more similar to the one described in use case I. Here both devices communicate using an intermediary. This intermediary is located on a different place, e.g. in a central server farm. This scenario would allow the use of available standards for recognizing the devices' abilities like WURFL. After identifying both devices, the server would be able to convert the information to a format that is suitable for the receiving device. This communication would be advanced, as it would allow more complex algorithms on a computational powerful server and it would also allow the connection of devices not being used in the same location. On the other hand, it does not completely fit the requirements of the research question, how to exchange details between small devices, using ontologies. It more describes a scenario on how to integrate any kind of devices, using a central server, over the internet. The devices also would require a internet connection, that allows communicating through the server. Some small devices might not have permanent internet connection. For this reason, the second communications channel will not be further discussed.

6.4 Ontological requirements

In this subchapter I will give an introduction to different needs the ontology describing the devices' capabilities would have to fulfill. Therefore I will first describe what parts are used in other protocols for exchanging this kind of information and afterwards add possible information not included in these protocols. Then I will discuss an example of structuring the ontology for supporting reasoning support, which will be described in chapter 8.5.

WURFL includes the following details about mobile devices:

- The name and version of the used device as well as further info on the pointing

method, keyboard support or the support of web sites.

- Details on supported mobile web standards specify the support of wml, chtml, etc.
- Details on the support of Ajax, flash lite and java script.
- The Browser cache goup specifies information on the use of the cache.
- The Display section specifies the format of the display.
- Supported image formats.
- The bugs section specifies wether basic authentication and HTML post is supported by the device.
- Security concerns are specified, like the support for https or wether the EMEI number of the device is accessible.
- The available video and audio playback is defined in the corresponding sections.
- The Support for rss and pdf is also specified.
- Message formats are specified, e.g. how to create a short link for sending a sms or Streaming capabilities define how the device is able to stream content.
- The Bearer group contains information on the maximum bandwidth of the device and on the support of WiFi. [Pas07]

For summarizing these descriptions: WURFL gives general information on devices, including their name, memory, telephone- and mobile messaging features. The main part of the descriptions is about the mobile internet features and the support for websites in general. It further describes the possibilities for the display of images, videos and audios, as well as for pdf files. The technical specifications for connecting to the internet are described by the maximum bandwidth and the ability of connecting to a WiFi Network. The whole aim of WURFL is to support mobile websites for different phones, so there are some abilities, related to other fields of mobile devices, not specified in the standard. [Pas07]

CC/PP is a more general standard that allows specifying new vocabulary following the ontological requirements defined in RDFS. New vocabulary can be introduced using XML Namespaces and the relationship to existing vocabulary can be specified using new RDFS statements. [KRW+04]

There are suggestions to plenty of different vocabulary for CC/PP: E.g. IETF media feature registration suggests the support for the description of different media types, e.g. the available fonts of a device, the different paper formats used by a printer, etc. This suggestion is more the suggestion of a syntax but offers some possible attributes as well. [HMH99]

WAP UAProf is a suggestion for a standard that describes the abilities of WAP browsers to a server and is very similar to WURFL, as both are based on the data provided by UAProf [WAPF01] There are further vocabularies suggested for the Audio Format Wave, MPEG 4, MPEG 7 and the printer working Groups' standards for printers [KRW+04].

Bandaras suggestion for an ontology [BPRC04] for the description of different devices will be used as a start for summarizing the required elements of an ontology. Figure 6.4 suggests possible content of an Ontology for our requirements.

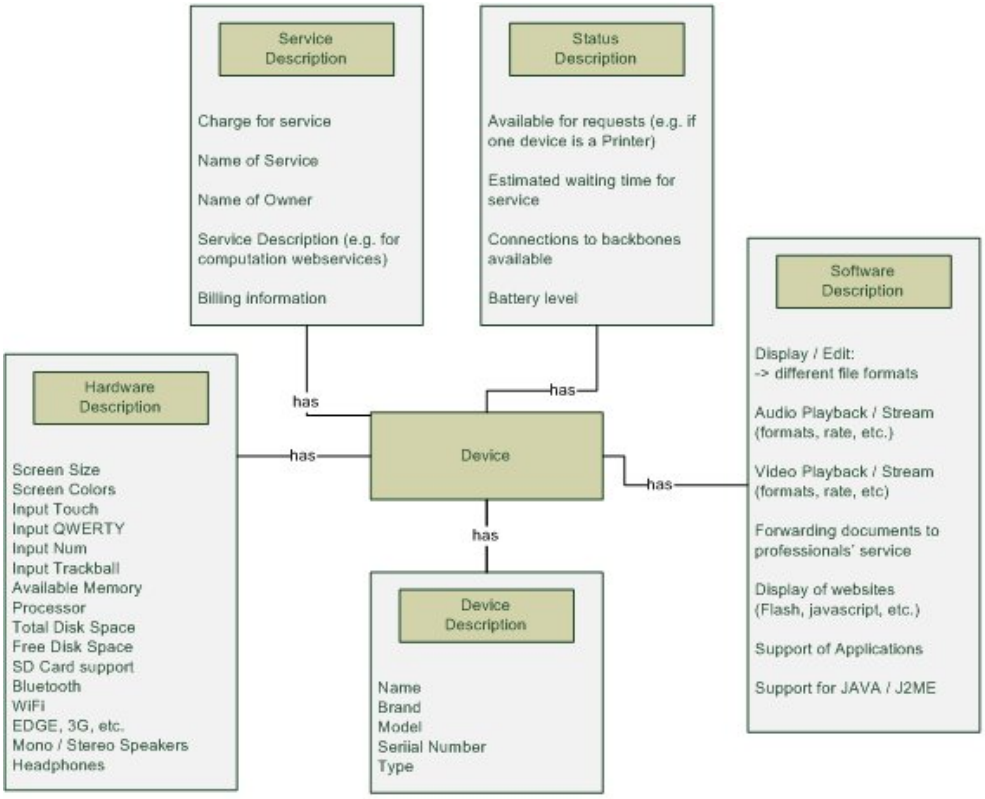

Figure 6.4: Possible Content for a device description, based on [BPRC04].

The hardware and software descriptions are oriented on WURFL, extended on some parts that seemed to be relevant to the author. The device description gives general information on the device. The Service Descriptions describe possible services, e.g. a

printer that charges a fee for printing a selected file. The status description is meant for describing the devices status, e.g. it could be currently not connected to the internet or the battery could be almost empty. This content is extendable, for describing all possible scenarios, it probably would require hundreds of pages. It is supposed to provide an overview, what elements are possible.

6.5 Reasoning Support

In this subchapter I will briefly discuss some basics on ontological reasoning. Afterwards I will describe software for reasoning support. Finally I will describe what basic reasoning support needs to be implemented for fulfilling the needs of the use case.

As described in chapter 2, ontologies are used for modeling a part of the real world, in a way, that allows a machine to compute this information. For allowing computation of the ontologies, the machines require some rules on what to compute.

An example would be an ontology about mobile devices, where is stated, that device A supports the Microsoft Office 2003 file format. Device B reads this ontology and wants to figure out, wether device A supports to recieve and display a PDF File and an XLS File. So it reads the ontology and tries to determine whether a PDF File is a MS Office 2003 file or not. The result will be no, as there is no PDF support specified in the ontology. Afterwards it will read the ontology again. There can be specified, that the xls file is a MS Office file. So it will rephrase the statement (device A, supports, Office 2003 Files) using the knowledge also provided by the ontology that:

(`Excel 2003 File, is_a, Office 2003 File`),

(`Word 2003 File, is_a, Office 2003 File`) etc. to

(`Device A, supports, Excel 2003 File`),

(`Device A, supports, Word 2003 file`), etc.

For allowing reasoning, agents require a knowledge base. This knowledge base is a formal representation of rules. Ontologies can be split into such rules by extracting all subjects predicate object statements. After having a rule base, reasoning applications try to solve problems by applying logic. [RN02] The above described example is one application that would allow the reasoning on a given knowledge base. For reasoning there are many implementations of artificial intelligence deduction calculi that follow different concepts and are used for different applications: ”General-purpose theorem proving and problem solving (first-order logic, simple type theory), program verification (first-order logic), distributed and concurrent systems (modal and temporal logics), program specification (intuitionistic logic), hardware verification (higher-order

logic), logic programming (Horn logic), and so on." [Por05] The differantiation of these algorithms are too complex for the scope of this thesis.

In the following I will briefly introduce some implementations for reasoning in a mobile context.

Wang et al. [WGZP04] have implemented a reasoning system that is based on the Jena semantic toolkit, described before. Their prototype uses a general ontology, which represents generally valid concepts, and an ontology representing context related rules. If the phone is located in the shower or the bedroom for example the low level context could be based on sensors that tell the users position, relating to a time, the user normally sleeps or takes a shower. This information could be mapped to a general ontology, providing the knowledge that if the user is sleeping or taking a shower, incoming calls shall be forwarded to the mailbox. If he is in the kitchen or his favorite nightclub, the ring tone level could be turned louder. There could be also a rule that all calls, received from persons tagged as business partners, get forwarded to the mailbox, if the user is at his girlfriends place after 6 pm and on weekends, but are patched through if he visits his parents in law on Sundays. These context aware ontologies allow the context aware reaction to events (e.g. incoming calls). [WGZP04]

Gu et al. [GPZ04] developed an ontology based, service oriented middleware for context aware services, that proposes an architecture for building and rapid prototyping of context-aware services. It is supposed to provide efficient support for acquiring, discovering, interpreting and accessing various contexts to build those services. They further propose an OWL based context model. They also use general ontology based reasoning, (e.g. if John is in his living room and the living room is part of his home, he is at home) and ontology reasoning (e.g. : "`(?user rdf:type socam:Person)`,
`(?user, socam:locatedIn, socam:BathRoom)`,
`socam:WaterHeater, socam:status, 'ON')`,
`(socam:BathRoom, socam:doorStatus, 'CLOSED')`
`-> (?user socam:status 'SHOWERING')]`" [GPZ04].

If the user is in his bathroom, the water heater is on and the door is locked, then he is showering.

For our application reasoning will be required for defining the different interactions possible. For example if the connected device does not support MS Word files, but PDF files, our device is able to convert MS Word to PDF, it would be possible to transfer a PDF file instead. If no conversion is possible, it is not possible to transfer anything. Another possible reasoning would be to stream a video, in case there are multiple versions, in the best resolution for a particular screen or an audio in mono or

stereo, depending on the speaker configuration of the specific device.

6.6 Suggestion of a system Architecture

In this subchapter I will summarize the previously discussed topics. I will then provide some thoughts on the implementation of a software that fulfills the needs, focusing on the capabilities of small devices.

In the previous subchapters I was providing information on existing implementations for exchanging devices abilities, Bluetooth as a communication standard for mobile devices, Ontologies for describing mobile devices, and reasoning support. In the system to design, first a connection between two devices will have to be established, afterwards information on the devices has to be exchanged, than the actual interaction can begin. Figure 6.5 illustrates these different Layers.

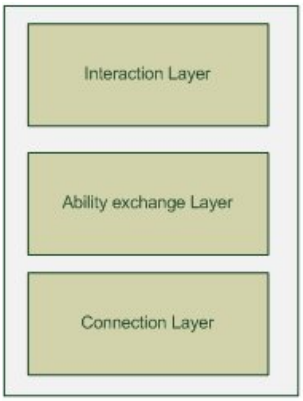

Figure 6.5: Layers of possible application.

The lowest layer is responsible for establishing a connection between two devices. The actual way of connecting the devices is not relevant to the exchange of information, a Bluetooth connection is possible and due to a lot of implemented interactions, an easy way of connecting. Connecting via WiFi or the internet is possible as well, but that would require the implementation of far more protocols, e.g. FTP for transferring files, streaming server and client for video or audio streaming, or proprietary protocols for the remote control of small medical devices. As Bluetooth is specifically designed for those local connections, it gives a lot of built in support for these interactions.

The ability exchange layer is responsible for exchanging the two devices abilities' both transferred in an OWL File. After exchanging, this layer is responsible for computing limitations to the possible interaction process, which can be done by comparing

the other devices abilities with its own abilities. In case the other device has restrictions, these functions have to be eliminated from the list of possible interactions. The ability exchange layer could also perform an analysis of possible conversions that can be performed for e.g. allowing additional file exchanges. The Interaction Layer receives a list of possible interactions and allows the user to select an interaction in a GUI.

Figure 6.6 presents an UML Sequence Diagram that illustrates the different steps in communication between two small devices.

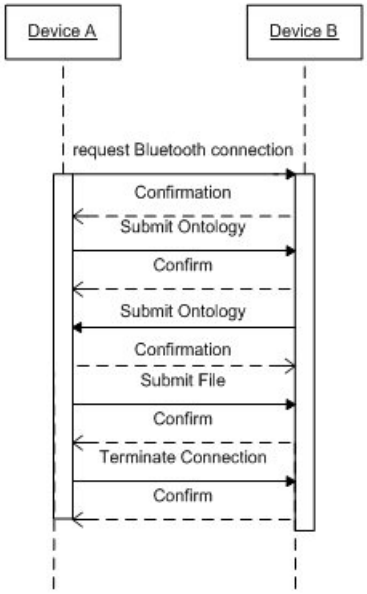

Figure 6.6: UML Sequence Diagram.

First a Bluetooth connection gets established. The protocol for that is far more complex, than described in the Figure, but for explaining the general behavior this simplification is sufficient. Afterwards both devices exchange their ontologies, and then information, described as compatible in the ontologies. Finally the connection gets terminated.

The UML Activity Diagram in figure 6.7 illustrates the activity of one of the small devices.

After establishing a Bluetooth connection both OWL Files containing the devices abilities get exchanged. Afterwards it is identified what abilities are supported for communications and what abilities can be emulated (e.g. format conversion word to PDF). Now the user can perform one or more actions and finally the connection gets terminated. The identification of possible communication channels gets performed by the reasoning engine of the application to develop, analyzing the devices abilities, the

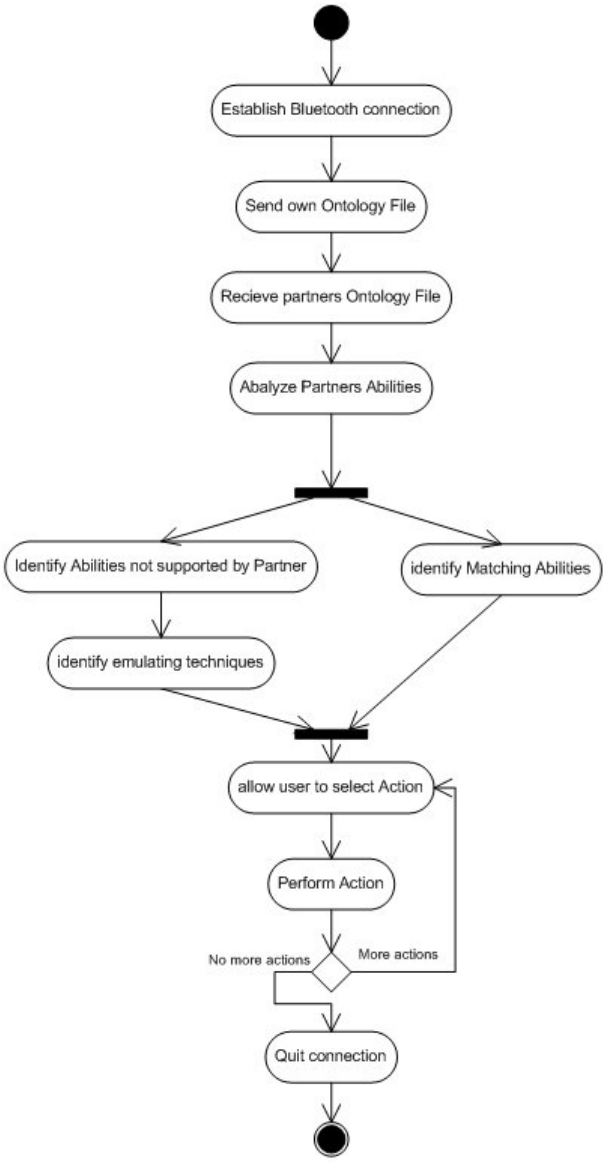

Figure 6.7: UML Activity Diagram.

partners' abilities and both devices abilities for format conversion. This analyze is performed using a rule base, that describes available formats, applications and documents. Further discussions, on what tasks have to be fulfilled for an implementation will be discussed in the conclusions and outlook chapter.

7 Conclusion and Outlook

In this chapter I summarizes the results of the research conducted in the thesis. I will compare the results to the initial research question and afterwards critically discuss the findings. After that I will compare the results of this thesis to existing findings in similar fields. I will finally give an outlook to research topics related to this thesis that could not be discussed sufficiently due to its limited focus.

The main goal of this thesis was to find out, how to allow users the use of different ontologies for tagging mobile generated content. Special considerations had to be made for the use of mobile devices with ontologies for tagging.

Therefore I was first giving a theoretical introduction to tagging and metadata. This introduction was important for the understanding of the research question. Afterwards I was briefly discussing the concepts of web services and service oriented architecture for giving the reader an introduction to those topics of the interoperability of devices and services.

Afterwards I was referring to ontologies as they are very important to the whole understanding of the thesis. I was summarizing different concepts and scenarios for the use of ontologies, like the semantic web. I was further describing the markup languages XML, RDF/ RDFS and OWL for the representation of ontologies. Finally I was demonstrating ways of sharing ontologies between different sources.

After presenting the general concepts of metadata, tagging and ontologies I was explaining the special circumstances of mobile devices. Here I was pointing out the diversity of those devices and later I was giving examples on special requirements for application design on mobile platforms. These explanations are very important for understanding the main parts of the thesis, as it was illustrating the need of an application development framework that is compatible to a wider range of devices. Therefore I was also explaining the main principles of web applications on mobile devices that later got used for developing an application for solving the research question of the thesis. These explanations were also very relevant for use case two that is concerned with the problem of exchanging different devices' abilities. Due to the diversity of mobile devices, these introductions gave an understanding to the need of having the exchange of

devices' abilities implemented in mobile devices for ensuring a better interoperability.

As the first chapters dealt with basic considerations, the next chapters tried to solve the problem of how it is possible to connect users on the one hand with professionals on the other hand using a mobile device.

In the process of requirements engineering, I was able to identify two different sides involved in the process of exchanging data: The users' and the professionals' side. The users' side is very divers. It can be e.g. a patient that wants so submit information to a doctor, a photographer that wants to submit a picture to his news-agencies' online picture shop or a sales employee that wants to submit a document to be correctly filed in the company's intranet. The only similarity is that all users are on the road and require mobile access to a professional's system.

There are also many different kinds of professionals on the other side: They have different systems, for the further computation of the submitted information: a Hospital Information System (in case of a patient-doctor communication), an online shop for news (in case of the journalist-agency communication) or a SAP system that is responsible for processing the sales person's document. The goal of the communication process is defining an interface that suits all different parties' needs.

After identifying the different actors involved in the process of submitting information, I was able to define two use cases: "Submit Information" and "Receive Information". The second use case requires the first one.

I defined the functional requirements of the system, illustrated in an UML Activity Diagram in Figure 3.5. The main non functional requirements consider usability and performance: It is very important to design a User Interface, where users are able to add the required metadata in as few steps as possible. They should be guided through the process of submitting information by a wizard that allows a clear orientation on the current point of the process. The performance is an important requirement as well, as information has to be submitted in two directions. First the user has to receive the ontology used for tagging; afterwards the selected tags and the information to submit have to be transferred to the receiving professionals' system. As small devices sometimes use slow internet connections, the requirement of performance has to be fulfilled by implementing an architecture that allows an advanced management of what kind of information is required on the device for tagging. On the one hand the user needs to know what tags he is able to use, on the other hand the limitations of slow internet connections and a small display limit the display of complex ontologies' structures. Applicable standards for the design of the software were identified that referred to the before described standards for ontologies, RDF/ RDFS and OWL.

These requirements were considered developing a specification for the implementation of an application for solving the goal of the research question.

For allowing different service providers with very different content to connect to different users, I defined a web service standard that has to be implemented by service providers. This web service allows a standardized connection, providing information about the service, the files and the meta data that can be submitted, as well as the ontology that has to be used for tagging and further descriptions of the service. This information is stored in WSDL for allowing a general description of different services. For discovery the services have to register at a service directory.

The System itself is split in two parts: One part on a web server and the other part actually running on the mobile device. Having these two different parts is necessary, considering the fact that ontologies used for tagging can be very complex and can have a large file size. Having the system split into a client- and a server side, allows having less data submitted to the mobile device. Therefore I defined services, running on the server side of the system, that submit the information required by the mobile device just in time. These services submit a small part of the ontology only, which is required by the user at the time of use. The limitations of the small devices screen also do not make it necessary to submit larger amounts of information at once, as the user is not able to display the whole amount of information.

These Services are able to submit different kinds of information. Using ontologies for tagging gives the user more abilities than just assigning words. As ontologies include relationships between different concepts, it is possible to browse through the available tags on an advanced level. The user can first select a class, he wants to use for tagging, e.g. a doctor. Afterwards he can view all properties related to that class, e.g. "`works_at`". Now he can browse the available values of the properties, e.g. "`Helsinki`" or "`Ilmenau`". The Implementation of a reasoning support could support the query by also offering names of states, even if the actual objects are just related to cities. If a part of the ontology would specify that e.g. "`Ilmenau`" is "`located_in`" "`Thuringia`", the system could also suggest "`Thuringia`" as a property. After selecting the desired value of the property the user could select the actual desired tag, e.g. "`Dr. Kaltwasser`", which is a "`doctor`", "`wors_at`", "`Ilmenau`".

The defined services provided by the server side of the system are able to submit all information required for the above described process, further they allow submitting the tagged information, receiving a list of available services and changing some user specific data.

One further advantage of using services and server side stored data is that users are

able to access their system from any kind of device. Further it is possible to easily reuse the services, e.g. for building a web interface for allowing the submission of data from any standard PC that provides internet access.

I also discussed the implementation of allowing the user to add concepts to an ontology. In the case of someone submitting a picture of the opponents' car in an accident to an insurance company, the user might want to introduce the new concept of the opponents' car. Therefore he needs to receive the required information for creating the new "car" object. He e.g. might need to enter the license plate and the brand of the car. I suggested submitting an OWL file to the small device, which gets interpreted by the small device's system, specifying the mandatory and facultative properties of a concept, for allowing the user to fulfill this task. The main concern about allowing the user to integrate new objects or even classes to an ontology is that ontologies are very complex constructs, that require a wide knowledge on the whole context for introducing concepts. If a user does not provide proper information in case of the opponents' car, the insurance would have to call him and demand more information. If the user adds a class, e.g. "motorbike" he might be able to think of the property "isA" "vehicle", but he might miss the property "hasSidewagon", as he might not think of it. He also might add wrong properties that are not related to the concept. This would weaken the ontology if some other user wants to reuse the concept "motorbike". As the concepts of the ontology shall be mapped to a professionals system, the professional would also have to map the new concepts, properties and objects to his system. The problem might be that some concepts are not compatible to the professionals system. For this reasons I did not recommended allowing the user to add new objects to existing ontologies.

I introduced a complex object model for the server side of the system that contains administrative objects for storing user information, communication objects that manage the communication to service providers and the service directory, a web service object, that is responsible for establishing communications to the small device and complex ontology objects that provide the functionality of browsing ontologies for classes, properties and objects. I finally recommended using an existing ontology API, like Jena, for providing the ontology related functionality. The object model on the client side is simple, as the client only needs to provide functions for accessing the server side logic.

The client side needs to provide the GUI. For this GUI I developed a wizard like interface that allows step by step inputs of the required information. Therefore I provided illustrations of the main Interfaces.

I developed a prototype for submitting information to a service provider. This pro-

totype is implemented using ASP .net, and provides a web application for the upload of information. This prototype implements the ONKI Selector toolset [TFVH09] that provides all ontology related tagging functionality. As the prototype has a demonstrative function only, I did not implement the use of concept based tag browsing. The prototype gives an impression of the "look and feel" of a web based application for the upload of data. Due to the lack of a professionals' System for receiving information I developed a web service that passes the submitted information to a web site which displays the data and the submitted metadata.

In a real life application it would be necessary to investigate professionals' systems, develop matching ontologies and integrate web services to those systems. This integration task would be system specific and would be an interesting research topic for different systems.

I suggested the use of web services for the client server connection, as they are implemented in most small devices' application development platforms. For testing the usability of my recommended system architecture, I would suggest the actual implementation of a system. Especially performance related concerns should be investigated by running tests of an application. The use of AJAX, e.g. in case of a web application, instead of using web services might be a different way for reaching the goal of implementing a fast connection from client to server platforms. This is also what was implemented in the demonstration application. Implementing CORBA support for more mobile devices might be another way of speeding up the connection. Unfortunately this topic is not very advanced yet. I discussed different technologies that allow the use of shared objects using Java J2ME and Windows Mobile. Those technologies are platform dependent, this was the reason for suggesting web services. Never the less, they might improve the connection speed of the different parts of the system. The defined web service functions are also aplicable to these other technologies, as the main functionality stays the same.

The implementation of the complete recommendations in a real world setting would also allow investigating the interoperability with real- life professionals' systems, and would allow qualified statements on possible savings for professionals and users. Depending on the necessary integration cost for professionals for developing an ontology that maps their specific requirements, I am convinced that the concept of connecting users to professionals for the submission of information would allow enormous savings of time and resources on the professionals' side. Also it would be easier for users to submit information in a standardized process that takes a maximum of three minutes, compared to submitting information in a classical manner, e.g. sending it by email.

Using advanced search options, smartly designed ontologies are also able to allow the user the use of tags he would not be able to think of by himself. Further investigating smart reasoning support could allow recommending the use of tags that might be relevant to the user, based on previous tagging by other users or the tags the user submitted before.

By provinding background information on the different basic technologies and suggesting an implementation for an application I was able to provide an example for an application that solves the research question. Never the less, it is just one possible application. There are other possible implementations, using different technologies or systems. I was trying to provide an approach with a wide interoperability. My research was able to connect the existing research on Ontologies and tagging to a mobile context, which was not provided yet.

Use Case two introduced a different way of using ontologies. Here I showed how ontologies can be used for exchanging small devices' abilities for allowing a communication based on their features. I discussed different devices on the market with a very broad spectrum of different abilities. I introduced different ways of describing devices' abilities and presented a suggestion for parts of an ontology for describing these features.

I described a possible procedure for the ontology based exchange of information on the devices that limits the possible communication to formats and standards supported by both devices. I briefly discussed the support of reasoning technologies for determining the possible interaction channels.

The research question, considering use case two, was how it is possible to use a similar technology as in use case one for exchaging details on mobile devices. Investigating this use case demonstrated, that it is very difficult to map the findings of use case one to use case two. I provided basic ideas for the exchange of mobile devices' features, and pointed out, that different technologies are required for this application.

Developing an application based on my suggestion, would require a very complex study of different abilities of small devices, the effects those differences have on communication processes, a reasoning algorithm that can determine the available interactions and the implementation of conversion algorithms for allowing further interoperability. This research topic would be worth a masters' thesis of its own, as the development of an application with reasoning support is a very complex and broad problem, that requires a wide knowledge on AI, the features of small devices and a study of relevant information, that could not be performed due to the limited scope of the second use case in this thesis.

Appendix

A Software Listing LifeManager Mobile Upload

A.1 MakeXML.cs

```
1
2 using System;
3 using System.Data;
4 using System.Configuration;
5 using System.Linq;
6 using System.Web;
7 using System.Web.Security;
8 using System.Web.UI;
9 using System.Web.UI.HtmlControls;
10 using System.Web.UI.WebControls;
11 using System.Web.UI.WebControls.WebParts;
12 using System.Xml.Linq;
13 using  System.Collections.Generic;
14 using System.ComponentModel;
15 using System.Xml;
16 using System.Xml.XPath;
17 using System.Globalization;
18 using System.IO;
19
20 //using CommonLib; // Suite class definition
21 //using InfoLib; // DisplayInfo() method
22
23
24 namespace LM_Mobile3
25 {
```

```
26      public class xmlobject
27      {
28
29          private DateTime _submitted_date = new DateTime();
30          public DateTime submitted_date
31          {
32              get { return _submitted_date; }
33              set
34              {
35                  if (value != _submitted_date)
36                  {
37                      _submitted_date = value;
38                      OnPropertyChanged("submitted_date");
39                  }
40              }
41
42          }
43
44          private string _used_vocabluary;
45          public string used_vocabluary
46          {
47              get { return _used_vocabluary; }
48              set
49              {
50                  if (value != _used_vocabluary)
51                  {
52                      _used_vocabluary = value;
53                      OnPropertyChanged("used_vocabluary");
54                  }
55              }
56
57          }
58          private string _to_service;
59          public string to_service
60          {
61              get { return _to_service; }
```

```
62              set
63              {
64                  if (value != _to_service)
65                  {
66                      _to_service = value;
67                      OnPropertyChanged("to_service");
68                  }
69              }
70
71          }
72          private string _doc_name;
73          public string doc_name
74          {
75              get { return _doc_name; }
76              set
77              {
78                  if (value != _doc_name)
79                  {
80                      _doc_name = value;
81                      OnPropertyChanged("doc_name");
82                  }
83              }
84
85          }
86          private string _doc_type;
87          public string doc_type
88          {
89              get { return _doc_type; }
90              set
91              {
92                  if (value != _doc_type)
93                  {
94                      _doc_type = value;
95                      OnPropertyChanged("doc_type");
96                  }
97              }
```

```
98
99          }
100
101         private string _user_name;
102         public string user_name
103         {
104             get { return _user_name; }
105             set
106             {
107                 if (value != _user_name)
108                 {
109                     _user_name = value;
110                     OnPropertyChanged("user_name");
111                 }
112             }
113
114         }
115         private string _first_name;
116         public string first_name
117         {
118             get { return _first_name; }
119             set
120             {
121                 if (value != _first_name)
122                 {
123                     _first_name = value;
124                     OnPropertyChanged("first_name");
125                 }
126             }
127
128         }
129         private string _last_name;
130         public string last_name
131         {
132             get { return _last_name; }
133             set
```

```
134             {
135                 if (value != _last_name)
136                 {
137                     _last_name = value;
138                     OnPropertyChanged("last_name");
139                 }
140             }
141 
142         }
143         private DateTime _birthday;
144         public DateTime birthday
145         {
146             get { return _birthday; }
147             set
148             {
149                 if (value != _birthday)
150                 {
151                     _birthday = value;
152                     OnPropertyChanged("birthday");
153                 }
154             }
155 
156         }
157         private string _address;
158         public string address
159         {
160             get { return _address; }
161             set
162             {
163                 if (value != _address)
164                 {
165                     _address = value;
166                     OnPropertyChanged("address");
167                 }
168             }
169 
```

```
170         }
171
172         private List<string> _tags = new List<string>();
173
174         public List<String> gettags()
175         {
176             return _tags;
177         }
178
179         public void addtag(String newtag)
180         {
181             if (!_tags.Contains(newtag))
182             {
183                 _tags.Add(newtag);
184
185             }
186         }
187
188
189         #region INotifyPropertyChanged Members
190
191         public void OnPropertyChanged(string Property)
192         {
193             if (PropertyChanged != null)
194             {
195                 PropertyChanged(this, new
                        PropertyChangedEventArgs(Property));
196             }
197         }
198
199         public event PropertyChangedEventHandler
            PropertyChanged;
200
201         #endregion
202
203
```

```
204         public Stream createXML(string folder, string filename
                )
205         {
206             string fileloc = folder + filename ;
207             XmlTextWriter writer = new XmlTextWriter((fileloc)
                    , null);
208             writer.WriteStartDocument();
209             writer.WriteStartElement("document");
210                 //statistical Data
211                 writer.WriteStartElement("statistical_data");
212                     //writer.WriteElementString(
213                     writer.WriteStartElement("submitted_date")
                            ;
214                     writer.WriteValue(submitted_date.ToString(
                            "yyyy/dd/MM_hh:mm:ss",CultureInfo.
                            InvariantCulture));
215                     writer.WriteEndElement();
216                     writer.WriteElementString("
                            used_vocablurary", used_vocabluary);
217                     writer.WriteElementString("to_service",
                            to_service);
218                     writer.WriteElementString("doc_name",
                            doc_name);
219                     writer.WriteElementString("doc_type",
                            doc_type);
220                 writer.WriteEndElement();
221                 //user_data
222                 writer.WriteStartElement("user_data");
223                     writer.WriteElementString("user_name",
                            user_name);
224                     writer.WriteStartElement("user_info");
225                         writer.WriteElementString("first_name"
                                , first_name);
226                         writer.WriteElementString("last_name",
                                 last_name);
227                         writer.WriteStartElement("birthday");
```

```
228                         writer.WriteValue(birthday.ToString("
                                yyyy/dd/MM",CultureInfo.
                                InvariantCulture));
229                         writer.WriteEndElement();
230                                                 writer.
                                                    WriteElementString
                                                    ("address",
                                                     address);
231                     writer.WriteEndElement();
232                 writer.WriteEndElement();
233                 //tags
234                 writer.WriteStartElement("tags");
235                     _tags.ForEach(delegate(String tagtemp)
236                     {
237                         writer.WriteElementString("tag",
                                tagtemp);
238                     });
239                 writer.WriteEndElement();
240             writer.WriteEndElement();
241             writer.WriteEndDocument();
242             writer.Close();
243
244             FileStream fs = new FileStream(fileloc,FileMode.
                    Open,FileAccess.Read);
245             return fs;
246         }
247
248         public void readxml(string xmlfile)
249         {
250             xmlfile = "c:\\temp\\xmlfile1.xml";
251             XPathDocument doc = new XPathDocument(xmlfile);
252             XPathNavigator nav = doc.CreateNavigator();
253             XPathExpression expr;
254
255             expr = nav.Compile("/document/statistical_data/
                    used_vocablurary");
```

```
256             XPathNodeIterator iterator = nav.Select(expr);
257             while (iterator.MoveNext())
258             {
259                 XPathNavigator nav2 = iterator.Current.Clone()
                        ;
260                 used_vocabluary = nav2.Value;
261             }
262
263             expr = nav.Compile("/document/statistical_data/
                    submitted_date");
264             iterator = nav.Select(expr);
265             while (iterator.MoveNext())
266             {
267                 XPathNavigator nav2 = iterator.Current.Clone()
                        ;
268                 submitted_date = Convert.ToDateTime(nav2.Value
                        );
269             }
270
271             expr = nav.Compile("/document/statistical_data/
                    to_service");
272             iterator = nav.Select(expr);
273             while (iterator.MoveNext())
274             {
275                 XPathNavigator nav2 = iterator.Current.Clone()
                        ;
276                 to_service = nav2.Value;
277             }
278
279
280             expr = nav.Compile("/document/statistical_data/
                    doc_name");
281             iterator = nav.Select(expr);
282             while (iterator.MoveNext())
283             {
284                 XPathNavigator nav2 = iterator.Current.Clone()
```

```
                ;
285             doc_name = nav2.Value;
286         }
287
288         expr = nav.Compile("/document/statistical_data/
                doc_type");
289         iterator = nav.Select(expr);
290         while (iterator.MoveNext())
291         {
292             XPathNavigator nav2 = iterator.Current.Clone()
                ;
293             doc_type = nav2.Value;
294         }
295
296         expr = nav.Compile("/document/user_data/user_name"
            );
297         iterator = nav.Select(expr);
298         while (iterator.MoveNext())
299         {
300             XPathNavigator nav2 = iterator.Current.Clone()
                ;
301             user_name = nav2.Value;
302         }
303
304         expr = nav.Compile("/document/user_data/user_info/
                first_name");
305         iterator = nav.Select(expr);
306         while (iterator.MoveNext())
307         {
308             XPathNavigator nav2 = iterator.Current.Clone()
                ;
309             first_name = nav2.Value;
310         }
311
312         expr = nav.Compile("/document/user_data/user_info/
                last_name");
```

```
313             iterator = nav.Select(expr);
314             while (iterator.MoveNext())
315             {
316                 XPathNavigator nav2 = iterator.Current.Clone()
                        ;
317                 last_name = nav2.Value;
318             }
319
320             expr = nav.Compile("/document/user_data/user_info/
                    birthday");
321             iterator = nav.Select(expr);
322             while (iterator.MoveNext())
323             {
324                 XPathNavigator nav2 = iterator.Current.Clone()
                        ;
325                 birthday = Convert.ToDateTime(nav2.Value);
326             }
327
328             expr = nav.Compile("/document/user_data/user_info/
                    address");
329             iterator = nav.Select(expr);
330             while (iterator.MoveNext())
331             {
332                 XPathNavigator nav2 = iterator.Current.Clone()
                        ;
333                 address = nav2.Value;
334             }
335
336             expr = nav.Compile("/document/tags/tag");
337             iterator = nav.Select(expr);
338             while (iterator.MoveNext())
339             {
340                 XPathNavigator nav2 = iterator.Current.Clone()
                        ;
341                 addtag(nav2.Value);
342             }
```

```
343
344
345                 //to_service = s.ChildNodes.Item(2).InnerText;
346                 //doc_name = s.ChildNodes.Item(3).InnerText;
347                 //doc_type = s.ChildNodes.Item(4).InnerText;
348
349                 //user_name = root.SelectSingleNode("user_name").
                        ToString();
350
351
352
353
354             }
355
356
357         }
358 }
```

A.2 ProviderModel.cs

```
1
2 using System;
3 using System.Data;
4 using System.Configuration;
5 using System.Linq;
6 using System.Web;
7 using System.Web.Security;
8 using System.Web.UI;
9 using System.Web.UI.HtmlControls;
10 using System.Web.UI.WebControls;
11 using System.Web.UI.WebControls.WebParts;
12 using System.Xml.Linq;
13 using System.ComponentModel;
14 using System.Collections.ObjectModel;
15 using System.IO;
16 using System.Text;
```

```
17
18 namespace LM_Mobile3
19 {
20     public class ProviderModel : INotifyPropertyChanged
21     {
22         private ObservableCollection<Provider> _providers;
23
24         public ProviderModel()
25         {
26         }
27
28         public ObservableCollection<Provider> Providers
29         {
30             get
31             {
32                 if (_providers == null)
33                 {
34                     _providers = new ObservableCollection<
                           Provider>();
35                     LoadData();
36                 }
37                 return _providers;
38             }
39         }
40
41         public ObservableCollection<Provider> GetData()
42         {
43             return _providers;
44         }
45
46         private void LoadData()
47         {
48             var xmlNodes = from element in XDocument.Load(File
                   .ReadAllText("C:\\Users\\nsommer.SOBERIT\\
                   Documents\\Visual Studio 2008\\Projects\\
                   LM_Mobile3\\LM_Mobile3\\App_Data\\provider.xml"
```

```
                ,Encoding.Default)).Elements("provider")
49                              select element;
50              foreach (var element in xmlNodes)
51              {
52                  Provider _provider = new Provider((string)
                        element.Attribute("id"),(string)element.
                        Attribute("name"),(string)element.Attribute
                        ("url"));
53                  _providers.Add(_provider);
54              }
55          }
56
57
58
59          #region INotifyPropertyChanged Members
60
61          public void OnPropertyChanged(string Property)
62          {
63              if (PropertyChanged != null)
64              {
65                  PropertyChanged(this, new
                        PropertyChangedEventArgs(Property));
66              }
67          }
68
69          public event PropertyChangedEventHandler
                PropertyChanged;
70
71          #endregion
72      }
73  }
```

A.3 Provider.cs

```
1
2  using System;
```

```
3  using System.Data;
4  using System.Configuration;
5  using System.Linq;
6  using System.Web;
7  using System.Web.Security;
8  using System.Web.UI;
9  using System.Web.UI.HtmlControls;
10 using System.Web.UI.WebControls;
11 using System.Web.UI.WebControls.WebParts;
12 using System.Xml.Linq;
13 using System.ComponentModel;
14 using System.Xml;
15 using System.Text;
16
17 namespace LM_Mobile3
18 {
19     public class Provider : INotifyPropertyChanged
20     {
21         public Provider(string id, string name, string url)
22         {
23             this.Id = id;
24             this.Name = name;
25             this.Url = url;
26         }
27
28         private string _id;
29
30         public string Id
31         {
32             get { return _id; }
33             private set
34             {
35                 if (value != _id)
36                 {
37                     _id = value;
38                     OnPropertyChanged("Id");
```

```
39                 }
40             }
41         }
42         private string _url;
43 
44         public string Url
45         {
46             get { return _url; }
47             private set
48             {
49                 if (value != _url)
50                 {
51                     _url = value;
52                     OnPropertyChanged("Url");
53                 }
54             }
55         }
56         private string _name;
57 
58         public string Name
59         {
60             get { return _name; }
61             private set
62             {
63                 if (value != _name)
64                 {
65                     _name = value;
66                     OnPropertyChanged("Name");
67                 }
68             }
69         }
70 
71         #region INotifyPropertyChanged Members
72 
73         public void OnPropertyChanged(string Property)
74         {
```

```
75              if (PropertyChanged != null)
76              {
77                  PropertyChanged(this, new
                        PropertyChangedEventArgs(Property));
78              }
79          }
80
81          public event PropertyChangedEventHandler
                PropertyChanged;
82
83
84
85          #endregion
86      }
87  }
```

A.4 SelectService.aspx

```
1
2
3  <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="
       selectservice.aspx.cs" Inherits="LM_Mobile3.WebForm2" %>
4
5  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
6
7  <html xmlns="http://www.w3.org/1999/xhtml" >
8  <head runat="server">
9  <script type="text/Javascript" src="http://www.yso.fi/onki.js
       ?0e82655d54d852e373e271378df1fe29&l="></script>
10 <script type="text/Javascript">
11 <!--
12 function addit(newval) {
13 document.form1.hidden1.value = document.form1.hidden1.value +
       newval;
14 }
```

```
15 -->
16 </script>
17     <title>Untitled Page</title>
18     <style type="text/css">
19
20
21         .style1
22         {
23             width: 147px;
24         }
25         #Button2
26         {
27             height: 26px;
28         }
29         </style>
30 </head>
31 <body>
32     <form id="form1" runat="server">
33     <div style="width: 450px">
34
35         <img alt="LifeManager mobile upload" src="lmhead.gif"
               /><br />
36  <asp:Label ID="Label3" runat="server" Text="Step 1 of 4:
     Select Service"
37             Visible="False"></asp:Label>
38         <br />
39
40         <asp:Panel ID="Panel4" runat="server">
41             <asp:Label ID="Label5" runat="server"
42                 Text="For simulating different users please
                       select one of the users first."></asp:Label
                      >
43             <br />
44             <br />
45             <asp:DropDownList ID="DropDownList2" runat="server
                   " Font-Size="Large"
```

```
46                  Width="224px">
47              </asp:DropDownList>
48              <br />
49              <br />
50              <asp:Button ID="Button9" runat="server" Height="50
                    px" onclick="Button9_Click"
51                  Text="Next␣-&gt;" />
52          </asp:Panel>
53
54          <br />
55          <asp:Panel ID="Panel1" runat="server" Visible="False">
56              <asp:Label ID="Label1" runat="server"
57      Text="Select␣your␣service␣here!" Visible="False"></asp:
            Label>
58              <br />
59              <asp:DropDownList ID="DropDownList1" runat="server
                    "
60                  onselectedindexchanged="
                        DropDownList1_SelectedIndexChanged" Height=
                        "33px"
61                  Width="224px" Font-Size="Large">
62              </asp:DropDownList>
63              <br />
64              <br />
65              <asp:Button ID="Button1" runat="server" onclick="
                    Button1_Click" Text="Next␣-&gt;"
66                  Width="77px" Height="50px" />
67              <asp:Button ID="Button7" runat="server" Height="50
                    px" onclick="Button7_Click"
68                  Text="Next␣-&gt;" Visible="False" />
69              <br />
70          </asp:Panel>
71          <asp:Panel ID="Panel3" runat="server" Visible="False">
72              Select your File to upload here!<br />
73              <table width="450">
74                  <tr>
```

```
75                      <td class="style1">
76                           Connected to:</td>
77                      <td>
78                          <asp:Label ID="connectedlabel0" runat=
                                "server" Text="ConnectedTo"></asp:
                                Label>
79                      </td>
80                  </tr>
81              </table>
82              <br />
83              <br />
84              <asp:FileUpload ID="FileUpload1" runat="server"
                    Height="40px" />
85              <br />
86              <br />
87              <asp:Button ID="Button3" runat="server" Height="50
                    px" onclick="Button3_Click"
88                  Text="Next␣-&gt;" />
89              <asp:Button ID="Button8" runat="server" Height="50
                    px" onclick="Button8_Click"
90                  Text="Next␣-&gt;" Visible="False" />
91              <br />
92          </asp:Panel>
93          <asp:Panel ID="Panel2" runat="server" Visible="False">
94              <table width="450">
95                  <tr>
96                      <td class="style1">
97                           Connected to:</td>
98                      <td>
99                          <asp:Label ID="connectedlabel" runat="
                                server" Text="ConnectedTo"></asp:
                                Label>
100                          
101                         <asp:Button ID="Button5" runat="server
                                " Height="25px" TabIndex="1"
102                             Text="change" onclick="
```

```
                                Button5_Click" Visible="False"
                                />
103                         <br />
104                     </td>
105                 </tr>
106                 <tr>
107                     <td class="style1">
108                          Selected File:</td>
109                     <td>
110                         <asp:Label ID="selectedLabel" runat="
                                server" Text="selectedLabel"></asp:
                                Label>
111                           
112                         <asp:Button ID="Button6" runat="server
                                " Height="25px" TabIndex="1"
113                                 Text="change" onclick="
                                    Button6_Click" Visible="False"
                                    />
114                     </td>
115                 </tr>
116                 <tr>
117                     <td class="style1">
118                          Tags:<br />
119                         <br />
120                         <br />
121                         Enter Tag:</td>
122                     <td>
123                         <input onkeyup="onki['mesh'].search({'
                                languageMenu': false, 'maxHits': 
                                10, 'onkimenu': false, '
                                openonkibutton': false, 'prefix': 
                                false, 'showOnkiName': false})"
                                id="meshSearch"/>
124                         <input type="hidden" id="uri" runat="
                                server"/>
125                         <input type="hidden" id="hidden1"
```

```
                                runat="server"/>
126                         </td>
127                     </tr>
128                     <tr>
129                         <td class="style1">
130                             <asp:Button ID="Button4" runat="server
                                    " Height="50px" onclick="
                                    Button4_Click"
131                                  TabIndex="1" Text="Next␣-&gt;" />
132                         </td>
133                         <td>
134                             <asp:Button ID="Button2" runat="server
                                    " Height="50px" onclick="
                                    Button2_Click"
135                                  Text="Submit" Visible="False" />
136                         </td>
137                     </tr>
138                 </table>
139             </asp:Panel>
140             <asp:Label ID="Label2" runat="server" Text="Label"
                    BorderStyle="None"
141                  ForeColor="Red"></asp:Label>
142             <br />
143             <asp:Label ID="Label4" runat="server" Text="Label"></
                    asp:Label>
144             <br />
145
146         </div>
147         </form>
148 </body>
149 </html>
```

A.5 SelectService.aspx.cs

```
1
2 using System;
```

```
3  using System.Collections;
4  using System.Configuration;
5  using System.Data;
6  using System.Linq;
7  using System.Web;
8  using System.Web.Security;
9  using System.Web.UI;
10 using System.Web.UI.HtmlControls;
11 using System.Web.UI.WebControls;
12 using System.Web.UI.WebControls.WebParts;
13 using System.Xml.Linq;
14 using System.Collections.Generic;
15 using System.Xml;
16 using System.IO;
17
18
19
20 namespace LM_Mobile3
21 {
22     //class provider : ListItem
23     //{
24     //     private string url;
25     //     public void set_url(string abc)
26     //     {
27     //          url = abc;
28     //     }
29     //     public void get_url()
30     //     {
31     //          return url;
32     //     }
33     //}
34
35     public partial class WebForm2 : System.Web.UI.Page
36     {
37         private ProviderModel _providerModel = new
               ProviderModel();
```

```
38          FileUpload fileup = new FileUpload();
39          ListItem anbieter = new ListItem();
40
41          protected void Page_Load(object sender, EventArgs e)
42          {
43              //this.DropDownList1.DataSource = _providerModel.
                    Providers;
44              //this.DropDownList1.DataTextField = "Name";
45              //this.DropDownList1.DataValueField = "Id";
46              //this.DropDownList1.DataBind();
47              if (DropDownList1.Items.Count < 1)
48              {
49                  ListItem li1 = new ListItem();
50                  ListItem li2 = new ListItem();
51                  li1.Text = "serviceprovider 1";
52                  li2.Text = "serviceprovider 2";
53                  li1.Value = "prov1";
54                  li2.Value = "prov2";
55                  DropDownList1.Items.Add(li1);
56                  DropDownList1.Items.Add(li2);
57              }
58              if (DropDownList2.Items.Count < 1)
59              {
60                  ListItem li3 = new ListItem();
61                  ListItem li4 = new ListItem();
62                  ListItem li5 = new ListItem();
63                  li3.Text = "Timo";
64                  li4.Text = "Matti";
65                  li5.Text = "Niko";
66                  li3.Value = "uid1";
67                  li4.Value = "uid2";
68                  li5.Value = "uid3";
69                  DropDownList2.Items.Add(li3);
70                  DropDownList2.Items.Add(li4);
71                  DropDownList2.Items.Add(li5);
72              }
```

```
73          }
74
75          protected void DropDownList1_SelectedIndexChanged(
                object sender, EventArgs e)
76          {
77
78          }
79
80          protected void Button1_Click(object sender, EventArgs
                e)
81          {
82              anbieter = DropDownList1.SelectedItem;
83              //if (anbieter != "")){
84              //     DropDownList1.SelectedItem = anbieter;}
85              Panel1.Visible = false;
86              Panel3.Visible = true;
87              connectedlabel.Text = (String)anbieter.Text;
88              connectedlabel0.Text = (String)anbieter.Text;
89              Label3.Text = "Step 2 of 4: Select File to upload.
                    ";
90
91          }
92
93          protected void Button2_Click(object sender, EventArgs
                e)
94          {
95              Panel2.Visible = false;
96              Label2.Visible = true;
97
98              //Label3.Text = "Step 2 of 2 Select File";
99              xmlobject xmlfile = new xmlobject();
100             xmlfile.submitted_date = DateTime.Now;
101             xmlfile.used_vocabluary = "Testvocabluary";
102             xmlfile.to_service = "Testservice";
103             xmlfile.doc_name = fileup.FileName.ToString(); ;
104             string fileext = fileup.FileName.ToString();
```

```
105
106             int posdot = fileext.LastIndexOf(".");
107             if (posdot >= 0)
108                 fileext = fileext.Remove(0, posdot);
109
110             xmlfile.doc_type =  fileext;
111
112             ListItem user = new ListItem();
113             user = DropDownList2.SelectedItem;
114             if ((String)DropDownList2.SelectedItem.Text == "
                    usr1")
115             {
116                 xmlfile.user_name = "MattiH";
117                 xmlfile.first_name = "Matti";
118                 xmlfile.last_name = "Hamalainen";
119                 xmlfile.birthday = Convert.ToDateTime("
                        22.04.1959");
120                 xmlfile.address = "SOBER␣IT␣Technikantie␣14,␣
                        02150␣Espoo,␣Finland";
121             }
122             if ((String)DropDownList2.SelectedItem.Text == "
                    usr2")
123             {
124                 xmlfile.user_name = "TimoI";
125                 xmlfile.first_name = "Timo";
126                 xmlfile.last_name = "Itala";
127                 xmlfile.birthday = Convert.ToDateTime("
                        24.1.1950");
128                 xmlfile.address = "SOBER␣IT␣Technikantie␣14,␣
                        02150␣Espoo,␣Finland";
129             }
130             if ((String)DropDownList2.SelectedItem.Text == "
                    usr3")
131
132             {
133                 xmlfile.user_name = "nsommer";
```

```
134             xmlfile.first_name = "niko";
135             xmlfile.last_name = "sommer";
136             xmlfile.birthday = Convert.ToDateTime("
                    14.09.1980");
137             xmlfile.address = "Servin␣Maijan␣Tie␣3A4,␣
                    02150␣Espoo,␣Finland";
138         }
139
140             //xmlfile.addtag("Tag1");
141         //xmlfile.addtag("Tag1");
142         //xmlfile.addtag("Tag2");
143         //xmlfile.addtag("Tag3");
144         //xmlfile.addtag("Tag4");
145
146         string tagstring = uri.Value.ToString();
147
148         if (tagstring != "")
149         {
150             int posraute = tagstring.IndexOf("#");
151             if (posraute >= 0)
152             tagstring = tagstring.Remove(0, posraute);
153         }
154             xmlfile.addtag(tagstring);
155         //Label4.Text = "posraute = " + Convert.ToString(
                posraute) + " tagstring = " + tagstring;
156
157         //xmlfile.readxml("asdf");
158             try
159             {
160                 string folder = Server.MapPath(".") + "/
                        App_Data/";
161                 Random random = new Random();
162                 string newbasename = Convert.ToString(
                        random.Next(10000, 999999999));
163                 string filename = newbasename + ".xml";
164
```

```
165                     var XMLStream = xmlfile.createXML(folder,
                            filename);
166
167                     var srXML = new BinaryReader(XMLStream);
168                     var srAttachment = new BinaryReader(fileup
                            .FileContent);
169                     //var srAttachment = new BinaryReader(
                            FileUpload1.FileContent);
170
171                     int lenXML = (int)srXML.BaseStream.Length;
172                     Byte[] bytearrayXML = new Byte[lenXML];
173                     srXML.Read(bytearrayXML, 0, lenXML);
174
175                     int lenAttachment = (int)srAttachment.
                            BaseStream.Length;
176                     Byte[] bytearrayAttachment = new Byte[
                            lenAttachment];
177                     srAttachment.Read(bytearrayAttachment, 0,
                            lenAttachment);
178
179                     de.sommernetz.www.Service1 service1 = new
                            LM_Mobile3.de.sommernetz.www.Service1()
                            ;
180                     service1.WriteLifeManagerEntry(filename,
                            bytearrayXML, fileup.FileName,
                            bytearrayAttachment);
181                     Label2.Text = "Your␣Information␣has␣been␣
                            submitted!";
182                 }
183             catch {
184
185                 Label2.Text = "There␣has␣been␣an␣error.␣Sorry.
                        ␣Please␣start␣again.";
186             };
187
188
```

```
189             //Label3.Text = onki_input.text ;
190             //Label4.Text = onki_field.Value.ToString();
191             //Label4.Text = uri.Value.ToString();
192             // Testbereich ende.
193         }
194
195         protected void Button3_Click(object sender, EventArgs
                e)
196         {
197             Panel3.Visible = false;
198             Panel2.Visible = true;
199             selectedLabel.Text = (string)FileUpload1.FileName;
200             Label3.Text = "Step␣3␣of␣4:␣Select␣your␣Tags.";
201
202             fileup = FileUpload1;
203             //Label4.Text = fileup.FileName.ToString();
204         }
205
206         protected void Button4_Click(object sender, EventArgs
                e)
207         {
208             Button4.Visible = false;
209             Button2.Visible = true;
210             Button5.Visible = true;
211             Button6.Visible = true;
212             Label3.Text = "Step␣4␣of␣4:␣Confirm␣or␣change␣
                    information.";
213             Label2.Text = uri.Value.ToString();
214         }
215
216         protected void Button5_Click(object sender, EventArgs
                e)
217         {
218             Panel2.Visible = false;
219             Panel1.Visible = true;
220             Button1.Visible = false;
```

```
221             Button7.Visible = true;
222
223         }
224
225         protected void Button6_Click(object sender, EventArgs
                e)
226         {
227             Panel3.Visible = true;
228             Panel2.Visible = false;
229             Button3.Visible = false;
230             Button8.Visible = true;
231         }
232
233         protected void Button7_Click(object sender, EventArgs
                e)
234         {
235             anbieter = DropDownList1.SelectedItem;
236             //if (anbieter != "")){
237             //     DropDownList1.SelectedItem = anbieter;}
238             Panel1.Visible = false;
239             Panel2.Visible = true;
240             connectedlabel.Text = (String)anbieter.Text;
241             connectedlabel0.Text = (String)anbieter.Text;
242         }
243
244         protected void Button8_Click(object sender, EventArgs
                e)
245         {
246             Panel3.Visible = false;
247             Panel2.Visible = true;
248             fileup = FileUpload1;
249             selectedLabel.Text = (string)fileup.FileName;
250
251         }
252
253         protected void Button9_Click(object sender, EventArgs
```

```
            e)
254         {
255             Panel4.Visible = false;
256             Panel1.Visible = true;
257             Label3.Visible = true;
258         }
259     }
260 }
```

B Introduction of OmaHyvivointi Project

”OmaHyvinvointi (stands for MyWellBeing) project is one of the Tekes Finnwell technology programs. It lasts from 1.1.2008 to 31.12.2009. The project is collaboration work between many universities and colleges: Helsinki University of Technology, Tempere University, Savonian Polytechnic from Kuopio, Kuopio University, Turku University and Abo Academi University. The main purposes of omaHyvinvointi project are:

- Create possibilities for citizen actively participate their own health information control.
- Produce the concept and the prototype of a citizen oriented system which integrate the needs of public and private
- Predict citizens future needs as the basis of the service concept
- Enable existing and new service providers to connect and to create new business opportunities.

To reach above goals, the project aim to develop a concept for a PHR system called Pärjäin. Pärjäin is a citizen survival kit, which provides a communication platform between service providers and citizen. The core function of Pärjäin is that citizen can collect, store and share their health information and documentation. Pärjäin might also have the following functions:

- Connect with other health device to monitor the citizen’s health condition
- Help citizen search and collect useful information about medication and health.
- Other functions like calendar and reminder, online booking the appointment with doctors etc.

The project will be divided into four themes, two common themes which are service conception and infrastructure and two specific case studies. From the common themes two common solutions should be produced. One is generic concept: How to make Pärjäin, which is citizens' needs, cost efficiently and scalable. What kind of system it should be and how to gain benefit and new business from it. The other one is generic infrastructure for service and information." [Han09]

Bibliography

[ACK04] Alonso, Gustavo ; Casati, Fabio ; Kuno, Harumi: *Web Services: Concepts, Architectures and Applications*. Berlin : Springer, 2004

[AH04] Antoniou, Grigoris ; Harmelen, Frank van: *A Semantic Web Primer*. Massachussetts : Massachussetts Institute of Technology, 2004

[AHMS06] Allen, Paul ; Higgins, Sam ; McRae, Paul ; Schlamann, Hermann: *Service Orientation: Winning Strategies and Best Practices*. Cambrige : Cambridge University Press, 2006

[App09a] Apple, w.A.: *Human Interface Principles: Creating a Great User Interface*. `http://developer.apple.com/iphone/library/documentation/UserExperience/Conceptual/MobileHIG/PrinciplesAndCharacteristics/PrinciplesAndCharacteristics.html#//apple_ref/doc/uid/TP40006556-CH7-SW1`, 2009

[app09b] apple, w.A.: *iPhone OS reference Library*. `http://developer.apple.com/iphone/library/navigation/index.html`, 2009

[app09c] apple, w.A.: *iPhone Technical Specifications*. `http://www.apple.com/iphone/specs.html`, 2009

[AS04] Alesso, Peter H. ; Smith, Craig F.: *Developing Semantic Web Services*. Wellesley : Peters, 2004

[Bes06] Best, Jo: *Analysis: What is a smart phone?* `http://www.silicon.com/technology/mobile/2006/02/13/analysis-what-is-a-smart-phone-39156391/`, 2006

[BLHL01] Berners-Lee, Tim ; Hendler, James ; Lassila, Ora: The Semantic Web. In: *The Scientific American* (2001)

[BLU07] BLUETOOTHDOC, w.A.: *Device Identification Profile Specification.* http://www.bluetooth.com/NR/rdonlyres/6F5750AC-8A3C-4A45-B8FD-BE5A4AEC921B/6544/DeviceID_SPEC_V13.pdf, 2007

[BMF06] Bieberstein, Norbert ; Marc Fiammante, Sanjay B.: *Service-Oriented Architecture Compass.* Upper Saddle River : Developer Works, 2006

[bmw08] bmwi, w.A.: *Verbreitung des Mobiltelefons in Familien oder bei Alleinerziehenden und bei Kindern.* http://www.itne.de/pdf/20070423182520-BMWi_-_Weitere_Stell.pdf, 2008

[BPRC04] Bandara1, Ayomi ; Payne1, Terry ; Roure1, David de ; Clemo2, Gary: *An Ontological Framework for Semantic Description of Devices.* eprints.ecs.soton.ac.uk/12689/1/DOPoster2.pdf, 2004

[Bur04] Burghadt, Markus: *Web Services.* Wiesbaden : Deutscher Universitäts Verlag, 2004

[Bus06] Bustarret, Eric: *Web Services and S60: Nokia releases a WSDL to C++ Wizard for S60.* http://www.newlc.com/Web-Services-and-S60-Nokia.html, June 2006

[But02] Butler, Mark H.: *Using capability classes to classify and match CC/PP And UAProf profiles.* http://www.hpl.hp.com/techreports/2002/HPL-2002-89.pdf, 2002

[BWB+02] Bultmann, Marion ; Wellbrock, Dr. R. ; Biermann, Heinz ; Engels, Jürgen ; Ernestus, Walter ; Wehrmann, Udo Höhn R. ; Schurig, Andreas: *Datenschutz und Telemedizin - Anforderungen an Medizinnetze -.* http://www.datenschutz-bayern.de/verwaltung/DatenschutzTelemedizin.pdf, 2002

[DT] Dev Team kxml: *kXML 2.* http://kxml.sourceforge.net/kxml2/,

[Ehr07] Ehrig, Marc: *Ontology Alignment - Bridging the Semantic Gap.* New York : Springer Science+Business Media, 2007

[Erl05] Erl, Thomas: *Service Oriented Architecture.* Prentice Hall Crawfordsville, 2005

[Esp05] Esposito, Dino: *Cookieless ASP.NET*. `http://msdn.microsoft.com/en-us/library/aa479314.aspx`, May 2005

[FFG+04] Foster, Ian ; Frey, Jeffrey ; Graham, Steve ; Tuecke, Steve ; Czajkowski, Karl ; Ferguson, Don ; Leymann, Frank ; Nally, Martin ; Sedukhin, Igor ; Snelling, David ; Storey, Tony ; Vambenepe, William ; Weerawarana, Sanjiva: *Modeling Stateful Resources with Web Services*. `http://www.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf`, 2004

[FGM05] Flissi, Areski ; Gransart, Christophe ; Merle, Philippe: A Component-based Software Infrastructure for Ubiquitous Computing. In: *ISPDC 05: Proceedings of the The 4th International Symposium on Parallel and Distributed Computing*. Washington, DC, USA : IEEE Computer Society, 2005. – ISBN 0–7695–2434–6, 183–190

[FHH04] Fikes, Richard ; Hayes, Patrick ; Horrocks, Ian: OWLQL A Language for Deductive Query Answering on the Semantic Web. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 2 (2004), December, Nr. 1, S. 19–29

[FLB+07] Fensel, Dieter ; Lausen, Holger ; Bruijn, Jos de ; Stollberg, Michael ; Roman, Dumitru: *Enabling Semantic Web Services*. Berlin/ Heidelberg : Springer, 2007

[GFF01] Gonçalves, Marcos A. ; France, Robert K. ; ; Fox, Edward A.: *Research and Advanced Technology for Digital Libraries*. Springer Berlin / Heidelberg, 2001

[GK09] Gessler, Ralf ; Krause, Thomas: *Wireless Netzwerke für den Nahbereich*. Vieweg und Teubner, 2009

[GPZ04] Gua, Tao ; Punga, Hung K. ; Zhang, Da Q.: A service oriented middleware for building context aware services. In: *Journal of Network and Computer Applications* (2004)

[GSHB05] Golder ; Scott ; Huberman ; Bernardo: The Structure of Collaborative Tagging Systems. In: *Hp Labs Technical report* (2005)

[Han09] Han, Yong: *Researchplan*. Helsinki, 2009

[Har04] HAROLD, Elliotte R.: *XML 1.1 Bible*. John Wiley & Sons, 2004

[HBB06] HURWITZ, Judith ; BLOOR, Robin ; BAROUDI, Carol: *Service Oriented Architecture for Dummies*. Hoboken : Wiley Publishing, 2006

[HMH99] HOLTMAN, K. ; MUTZ, A. ; HARDIE, T.: *Media Feature Tag Registration Procedure*. http://www.faqs.org/ftp/rfc/pdf/rfc2506.txt.pdf, 1999

[HTC09] HTC, w.A.: *HTC Touch Pro2 Specification*. http://www.htc.com/www/product/touchpro2/specification.html, 2009

[int08] INTERPRETINGTECH, w.A.: *Opera Mobile 9.5 with Flash Lite 3.0*. http://www.interpreting-tech.com/bemobile/?p=107, 2008

[IRRH03] INDULSKA, Jadwiga ; ROBINSON, Ricky ; RAKOTONIRAINY, Andry ; HENRICKSEN, Karen: *Lecture Notes in Computer Science Mobile Data Management*. Berlin : Springer, 2003

[Jac95] JACKSON, Michael: *Software Requirements and Specifications*. London : Addison Wesley, 1995

[Jena] JENADEVTEAM, w.A.: *Jena A Semantic Web Framework for Java*. http://jena.sourceforge.net/,

[Jenb] JENADEVTEAM, w.A.: *Jena Ontology API*. http://jena.sourceforge.net/ontology/,

[Jin09] JINI, w.A.: *Jini.org Main Page*. http://www.jini.org/wiki/Main_Page, December 2009

[Kam98] KAMADA, Tomihisa: *Compact HTML for Small Information Appliances*. http://www.w3.org/TR/1998/NOTE-compactHTML-19980209/#www3, 1998

[Kec09] KECKER, Christoph: *UML 2: Das umfassende Handbuch*. Galileo Press, 2009

[Kei08] KEIZER, Gregg: *Windows market share dives below 90% for first time*. http://www.computerworld.com/s/article/9121938/Windows_market_share_dives_below_90_for_first_time, 2008

[Kru98] KRUCHTEN, Philippe: *The Rational Unified Process. An Introduction*. Adisson Wesley, 1998

[KRW+04] Klyne, Graham ; Reynolds, Franklin ; Woodrow, Chris ; Ohto, Hidetaka ; Hjelm, Johan ; Butler, Mark H. ; Tran, Luu: *Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0 W3C Recommendation 15 January 2004*. http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/, 2004

[Kue03] Kuester, Mark W.: Web Services - Versprechen und Realität. In: *Praxis der Wirtschaftsinformatik* (2003), December

[Kwa99] Kwasnik, Barbara H.: The Role of Classification in in Knowledge Representation and Discovery. In: *Library Trends* (1999)

[LVSC03] Lyman, Peter ; Varian, Hal R. ; Swearingen, Kirsten ; Charles, Peter: How much information 2003. In: *School of Information Management and Systems, University of California at Berkeley* (2003)

[LW00] Leffingwell, Dean ; Widrig, Don: *Managing Software Requirements - A Unified Approach*. Addison Wesley, 2000

[Mah03] Mahmoud, Qusay H.: *Part II: The Java APIs for Bluetooth Wireless Technology*. http://developers.sun.com/mobility/midp/articles/bluetooth2/, 2003

[Mat04] Mathes, Adam: Folksonomies Cooperative Classification and Communication Through Shared Metadata. In: *Computer Mediated Communication (LIS590CMC)* (2004), 12

[Mic09] Microsoft, w.A.: *Windows Mobile 6.1*. http://www.microsoft.com/windowsmobile/en-us/meet/communications.mspx, 2009

[Mid] Middsol, w.A.: *MinCor.NET*. http://www.middsol.de/mincor.htm,

[MNbD06] Marlow, Cameron ; Naaman, Mor ; boyd danah ; Davis, Marc: HT06, Tagging Paper, Taxonomy, Flickr, Academic Article,ToRead. In: *ACM Press* (2006), 8

[MRM+08] Marín, Ignacio ; Rionda, Abel ; Martinez, David ; Montes, Celia ; Pedrosa, Germán ; Campos, Antonio: NOMADIC DEVICE IDENTIFICATION AND CLIENT PROVISION FOR INTERACTION IN A VEHICULAR NETWORK. In: *IADIS International Conference Wireless*

Applications and Computing 2008. http://www.iadis.net/dl/final_uploads/200807L004.pdf, 2008

[msd08] MSDN, w. A.: *Cookieless ASP.NET*. http://msdn.microsoft.com/en-us/library/aa479314.aspx, August 2008

[msd09] MSDN, w.A.: *Getting Started in Developing Applications for Windows Mobile 6*. http://msdn.microsoft.com/en-us/library/bb158522.aspx, 2009

[Nag06] NAGEL, Sebastian: *Morphologie und Lexikographie*. http://www.cis.uni-muenchen.de/~wastl/kurse/morpho/AB15.pdf, 2006

[OAS05] OASIS, w. A.: *OASIS UDDI Specification TC*. http://www.oasis-open.org/committees/uddi-spec/, February 2005

[onJ02] ONJAVA, w.A.: *j2me*. http://onjava.com/onjava/2002/03/27/j2me.html, March 2002

[OP02] OSTERWALDER, Alexander ; PIGNEUR, Yves: *An eBusiness Model Ontology for Modeling eBusiness*. http://www.bookpoint.co.th/bookpoint/spaw2/uploads/files/0202004.pdf, 2002

[Ope01] OPENWAVESYSTEMSINC, w.A.: *WML 1.3 Developer's Guide*. http://developer.openwave.com/docs/51/wml_dev_guide.pdf, 2001

[Ope09] OPERA, w.A.: *Opera Mobile benefits*. http://www.opera.com/business/solutions/mobile/benefits/, 2009

[Pas05] PASHTAN, Ariel: *Mobile Web Services*. Cambridge University Press, 2005

[Pas07] PASSANI, Luca: *WURFL Devices and WURFL Capabilities*. http://wurfl.sourceforge.net/help_doc.php#chtml_ui, 2007

[Pas08] PASSANI, Luca: *Introducing WALL: a Library to Multiserve Applications on the Wireless Web*. http://wurfl.sourceforge.net/java/tutorial.php, 2008

[Pir02] PIROUMIAN, Vartan: *Wireless J2ME Platform Programming*. Palo Alto : Sun Microsystems, Inc, 2002

[Por05] Portoraro, Frederic: *Automated Reasoning Stanford Encyclopedia of Philosophy.* http://plato.stanford.edu/entries/reasoning-automated/, 2005

[Pre05] Pressman, Roger S.: *Software Engineering, A Practitioner's Approach.* Singapore : McGraw-Hill, 2005

[RN02] Russell, S.J. ; Norvig, P.: *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2002

[Sad08] Sadun, Erica: *Creating an iPhone-based Web Service: Part 1.* http://blogs.oreilly.com/iphone/2008/09/creating-an-iphone-based-web-s.html, September 2008

[Sch03] Schoening, James: *IEEE P1600.1 Standard Upper Ontology Working Group.* http://suo.ieee.org/, 2003

[Som04] Sommerville, Ian: *Software Engineering.* Harlow : Addison Wesley, 2004

[SUN08] SUN, w.A.: *Sun Java Wireless Client Available On Latest HP iPAQ Smartphone.* http://www.sun.com/aboutsun/pr/2008-07/sunflash.20080707.1.xml. Version: 2008

[Tay99] Taylor, Arlene G.: *The Organization of Information.* Englewood : Libraries Unlimited, 1999

[TFVH09] Tuominen, Jouni ; Frosterus, Matias ; Viljanen, Kim ; Hyvonen, Eero: ONKI SKOS Publishing and Utilizing Thesauri in the SemanticWeb. In: *Proceedings of the 6th European Semantic Web Conference* (2009)

[Vos03] Voss, Jakob: *Begriffssysteme. Ein Vergleich verschiedener Arten von Begriffssystemen und Entwurf des integrierenden Thema-Datenmodells.* www.jakobvoss.de/epub/begriffssysteme03/begriffssysteme.pdf, 12 2003

[Vos07] Voss, Jakob: *Tagging, Folksonomy and Co. Renaissance of Manual Indexing?* http://arxiv.org/abs/cs.IR/0701072, Jan 2007. – Submitted to the 10th International Symposium for Information Science Cologne

[WAPF01] Wireless Application Protocol Forum, w.A.: *WAG UAProf*. http://www.openmobilealliance.org/tech/affiliates/wap/wap-248-uaprof-20011020-a.pdf, 2001

[WGZP04] Wang, Xiao H. ; Gu, Tao ; Zhang, Da Q. ; Pung, Hung K.: Ontology Based Context Modeling and Reasoning using OWL. In: *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference* (2004)

[Wue05] Wuest, Bjoern: *Framework for middleware executed on mobile devices*. http://deposit.ddb.de/cgi-bin/dokserv?idn=986624551, UNI Kassel, Diss., 2005

[YJNT07] Yanbe, Yusuke ; Jatowt, Adam ; Nakamura, Satoshi ; Tanaka, Katsumi: *Can social bookmarking enhance search in the web?* http://dx.doi.org/10.1145/1255175.1255198. Version: 2007

[ZHA05] ZHANG, ZHIJUN: *ONTOLOGY QUERY LANGUAGES FOR THE SEMANTIC WEB: A PERFORMANCE EVALUATION*. http://www.cs.uga.edu/~jam/home/theses/zhijun_thesis/final/zhang_zhijun_200508_ms.pdf, University of Georgia, Diplomarbeit, 2005

List of Figures

List of Abbreviations

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CHTML	Compact HTML
CORBA	Client Object Request Broker
ESB	Enterprise Service Bus
GPS	Global Positioning System
HTML	Hypertext Markup Language
HTTPS	Hypertext Transfer Protocol Secure
I/O	Input/ Output
IP	Internet Protocol
OPAC	Online Public Access Catalog
OWL	Web Ontology Language
RDFS	Resource Description Framework Schema
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
SRS	Software Requirements Specification
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UML	Unified Modeling Language
URI	Uniform Resource Identifier
WML	Wireless Markup Language
WSDL	Web Service Description Language
WURFL	Wireless Universal Resource FiLe
XHTML	Extensible Hyper Text Markup Language
XML	Extensible Markup Language

Theses of this Diplomarbeit

1. Using ontologies for tagging mobile uploaded content allows the further preprocessing of the submitted content at the service srovider.
2. Using ontologies for tagging allows the user a new tagging method that can support him finding appropriate tags by browsing the used ontology.
3. Due to the size of large ontologies, the transfer of ontologies to mobile devices has to be split into parts, submitting the information only when required.
4. The User Interface for browsing an ontology has to be implemented in an easy to use way, that allows the user the selection of tags in a few, simple steps.
5. Due to the diversity of small devices, they require a description of their features for ensuring interoperability.
6. Using ontologies for describing small devices allows the implementation of Reasoning Support for a simple detection of possible communication channels and for ensuring interoperability.

Wiesbaden, den 18. 12. 2009 Nikolaus Daniel Sommer

Erklärung

Die vorliegende Arbeit habe ich selbstständig ohne Benutzung anderer als der angegebenen Quellen angefertigt. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Quellen entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer oder anderer Prüfungen noch nicht vorgelegt worden.

Ilmenau, den 31. 12. 2004 Nikolaus Daniel Sommer